Éléments de flexibilité et d'efficacité en programmation par contraintes

Xavier Lorca

Éléments de flexibilité et d'efficacité en programmation par contraintes

HABILITATION À DIRIGER LES RECHERCHES présentée par :

Xavier LORCA, M.A., équipe TASC,
École des Mines de Nantes, Université de Nantes, Inria.

Le 6 novembre 2014.

Rapporteurs :

Christian BESSIÈRE, Dir. de Recherche, équipe COCONUT, Montpellier.
Jean-Charles RÉGIN, Pr. Université, équipe CeP, Sophia Antipolis.
Gilles PESANT, Pr. Université de Montréal.

Examinateurs :

Nicolas BELDICEANU, Pr. Mines Nantes, équipe TASC, Nantes.
Pierre FLENER, Pr. Université Uppsala, Suède.
Narendra JUSSIEN, Pr. Inst. Mines-Télécom, Dir. Télécom Lille.
Éric MONFROY, Pr. Université de Nantes, équipe TASC, Nantes.

Table des matières

Chapitre 1

Contexte de cette étude

Historiquement, la programmation par contraintes [MAC 77, HEN 89] est une discipline située à la croisée de nombreux domaines comme la recherche opérationnelle, l'analyse numérique, le calcul symbolique, la programmation logique (Prolog [COL 90]) et l'intelligence artificielle (ALICE [LAU 78]). Elle apporte une nouvelle approche à la résolution de problèmes combinatoires en tentant de combiner et d'unifier le meilleur de chacune de ces disciplines. Les premiers travaux dans ce domaine remontent aux recherches effectuées à la frontière entre l'intelligence artificielle et l'infographie [MON 74] dans les années 1970. Durant les deux dernières décennies, une prise de conscience des enjeux relatifs à l'existence d'un paradigme déclaratif, permettant de modéliser, d'implémenter et résoudre des problèmes combinatoires, s'est opérée. Ainsi, l'unification des différentes approches proposées autour de la résolution de problèmes combinatoires complexes a donné lieu à l'émergence d'un cadre commun tant conceptuel que pratique : la programmation par contraintes. Notons que les industriels, tels que Bull, Cosytec, IBM, ICL, Ilog, Prologia, Siemens ou Renault, ont été les premiers à réaliser l'intérêt pratique de la programmation par contraintes dans la résolution et/ou l'optimisation de problèmes combinatoires réels.

Raisonner par "contraintes" c'est s'interroger sur les *propriétés intrinsèques* caractérisant la sémantique d'un problème. Nous entendons par là que cette discipline a pour vocation de s'abstraire autant que possible de tout *raisonnement opérationnel* afin de se concentrer sur une spécification conceptuelle de chaque problème. En cela, la programmation par contraintes est un *paradigme déclaratif*. Cette séparation des préoccupations offre une flexibilité remarquable dans le monde de l'intelligence artificielle et de la recherche opérationnelle. On pourra aisément exprimer des contraintes non linéaires pour un problème, par exemple forcer deux variables à prendre des valeurs distinctes. Cependant, cette flexibilité se paie généralement par la rigueur et l'expertise nécessaire lorsque l'on va s'intéresser à la mise en œuvre (donc au caractère opérationnel) de ce type de raisonnement. De manière très classique, nous réduirons, tout au long de ce manuscrit, notre propos au cas des systèmes de programmation par contraintes basés sur une représentation des domaines finis. La mise en œuvre d'un système de programmation par contraintes commence par une architecture pragmatique et rationnelle des services nécessaires à la conception d'un outil flexible et robuste. Aujourd'hui, tout système de programmation par contraintes se doit de proposer trois composantes essentielles :

1) *Propagation* : mécanisme systématique permettant de détecter et supprimer les parties inconsistantes des domaines des variables, c.-à-d. les valeurs ne pouvant apparaître dans aucune solution ;

2) *Recherche* : de l'affectation variable/valeurs pour les problèmes de décision, jusqu'à la recherche par voisinage large (LNS) pour les problèmes d'optimisation ;

3) *Modélisation* : offrir, à un utilisateur, un accès simple, expressif mais néanmoins complet à l'outil, c.-à-d. de la représentation des variables, domaines et contraintes jusqu'à la définition de certains paramètres opérationnels de l'outil, typiquement la configuration de la recherche.

Les deux premiers points sont l'essence des travaux liés à l'efficacité dans les systèmes de programmation par contraintes. On y retrouve toutes les contributions liées aux algorithmes de propagation qui forment les bases des problèmes de satisfaction de contraintes (CSP), ainsi que toutes les avancées liées aux stratégies de recherches (complètes ou incomplètes) ou aux heuristiques de branchement dans la stratégie choisie, qu'elles soient dédiées à un problème ou génériques, c.-à-d., liées à la structure du réseau de contraintes ou à des statistiques dynamiques sur l'état des domaines, des variables et des contraintes. Pour un état de l'art plus précis et complet, le lecteur pourra se référer à [SCH 06] et [PES 12] qui offrent une description fidèle de l'équilibre aujourd'hui nécessaire à la conception d'un outil réaliste en programmation par contraintes, à la fois sur le volet propagation et sur le volet recherche. Cependant, le souci d'efficacité ne doit pas occulter le côté flexibilité, élément différenciant de la programmation par contraintes. C'est en effet ce dernier qui permet aux systèmes d'évoluer proprement et sereinement au fil des avancées de la communauté mais aussi et surtout qui assure aux utilisateurs une prise en main (autant que faire se peut) aisée du paradigme. Ainsi, le dernier point constitue la majeure partie du travail lié à la flexibilité dans les systèmes car il a fait l'objet de nombreuses publications et il n'existe pas de modèle dominant à l'exception, peut-être, du langage MiniZinc [NET 07].[1] L'ouverture, au niveau de la modélisation, de la stratégie de recherche est quelque part un aveu d'impuissance pour un paradigme purement déclaratif, mais a le mérite de soulever une des questions que nous nous poserons dans la suite : Peut-on ouvrir la configuration de la stratégie de propagation au niveau de la modélisation ?

1. `http://www.minizinc.org/`

Aujourd'hui parler de flexibilité et d'efficacité en programmation par contraintes c'est s'interroger sur la part qu'occupe chacun des éléments du triptyque - Modélisation, Recherche, Propagation - du point de vue d'un utilisateur. Idéalement, ce dernier ne souhaite pas dépasser le stade des considérations de modélisation et c'est aussi à ce niveau que les gains de performances sont les plus importants entre deux modèles d'un même problème. Pourtant, le paradigme purement déclaratif, cher à l'Intelligence Artificielle, reste aujourd'hui encore un vœu pieux. En effet, la situation se complique dès que l'on dépasse le cadre des problèmes de satisfaction et que l'on considère les problèmes d'optimisation. En effet, l'utilisateur doit s'interroger sur l'adaptation de la stratégie de résolution à ses besoins : Son problème est-il de satisfaction ou d'optimisation ? Dans le dernier cas l'optimalité est-elle requise ? A-t-il des contraintes sur le temps de réponse ? Ceci est aujourd'hui une problématique bien abordée en programmation par contraintes pour ce qui concerne la description de stratégies de résolution [HAR 79, BES 96, SMI 98] (stratégie de recherche ou heuristiques de branchement). Cela se complique réellement dans le cadre des recherches incomplètes à base de méta-heuristiques où une paramétrisation fine est nécessaire et peu compatible dans une intégration au niveau du modèle ; il s'agit donc d'une limite de la flexibilité des systèmes actuels. Dans ce paysage, le volet propagation reste, jusqu'à aujourd'hui, monolithique dans le cadre des systèmes de programmation par contraintes. En effet, autant de nombreux travaux ont porté sur des formes diverses d'algorithmes de propagation [RÉG 04, BES 06], autant les systèmes modernes n'offrent aucune forme de flexibilité ou de composition à ce niveau, souvent dans le but avoué de préserver les performances. Même si l'espérance de gain est modeste,[2] tout reste donc à faire dans ce domaine afin d'offrir à l'utilisateur une possibilité d'adapter, autant que faire ce peut, le fonctionnement de son algorithme de propagation à son problème, sans pour autant connaître finement la mécanique interne du système de contraintes sur lequel il travaille.

1.1. Positionnement de l'étude

L'objet de ce document est donc de synthétiser différents éléments et questionnements autour d'aspects liés à la propagation, aux filtrages et à la stratégie de recherche

2. les algorithmes de propagation, mis en œuvre dans les systèmes de contraintes actuels, sont souvent polynomiaux en la taille du problème considéré.

dans les systèmes de contraintes. Plus précisément, mon intérêt s'est porté sur l'analyse des stratégies de propagation dans les systèmes de contraintes, le comportement des algorithmes de filtrages au sein de ces stratégies de propagation, mais aussi la prise en compte de besoin métiers de plus en plus pressants autour de stratégies de recherche pour les problèmes d'optimisation. Le lien entre ces éléments a été porté par une interrogation permanente sur la dichotomie entre une vision très conceptuelle de la programmation par contraintes, reposant sur la notion de déclarativité (l'outil de résolution étant alors entièrement caché à l'utilisateur), et une vision très opérationnelle où l'utilisateur final souhaite maitriser parfaitement son processus de résolution : Quelle stratégie de recherche et pour quel problème ? Quelle contrainte / filtrage utiliser et dans quel contexte ? L'optimalité est-elle un besoin ou la recherche d'une solution de bonne qualité est-elle suffisante ?

Recruté en octobre 2008 en tant que maître assistant à l'École des Mines de Nantes, mon intégration reposait sur l'évolution de l'outil Choco développé au sein de l'École des Mines de Nantes depuis 1999 [LAB 00]. L'équipe de développeurs à cette date (Charles Prud'Homme, Narendra Jussien et moi-même) constatait trois lacunes dans la gestion initiale et le développement de l'outil. Elles peuvent se résumer ainsi :

– un outil pour l'enseignement et la recherche développé par des chercheurs au fil de leurs besoins, conduisant à peu de cohérence dans le développement du système ;

– des algorithmes de filtrage dans les contraintes (globales) très performants et en pointe de l'état de l'art, mais un moteur de propagation et de recherche cruellement en retard par rapport aux dernières avancées de la communauté [TEA 13b] ;

– un outil à visibilité limitée au sein de la communauté et des industriels.

De manière pragmatique, une stratégie en trois temps, mêlant ingénierie et recherche, avec l'objectif de promouvoir l'outil, s'est mise en place : (1) centralisation et capitalisation de la connaissance ; (2) mise à jour de l'outil avec les interfaces de modélisation de la communauté : séparation de la couche modélisation et du cœur opérationnel du système, implémentation de référence pour le JSR-331[3] ; (3) refonte du cœur de l'outil dans le respect de l'esprit initial : un outil de recherche pour la recherche et

3. JSR, pour *Java Specification Request*, est un système normalisé permettant de définir des évolutions de la plate-forme Java. Le JSR-331 défini une API Java dédiée à la programmation par contraintes.

l'enseignement combinant flexibilité et efficacité. Ce dernier point constitue la trame de ce document qui va synthétiser les résultats obtenus durant ces cinq dernières années et a fourni des résultats très satisfaisants (Compétition MiniZinc 2013, ANR *laboratoire commun* avec la société EuroDécision, Google grant).

Les travaux présentés ici reposent sur des collaborations diverses et variées tant au niveau local (Nicolas Beldiceanu, Rémi Douence, Danièle Gardy, Narendra Jussien, Thierry Petit, Charlotte Truchet) qu'au niveau international (Pierre Flener, Irit Katriel, Louis-Martin Rousseau). Ces collaborations et travaux m'ont conduit à encadrer deux thèses, celle de Charles Prud'Homme (initialement ingénieur d'étude en charge de l'outil Choco, puis doctorant) et celle de Jean-Guillaume Fages. Les travaux rapportés dans le présent document ont donc fait l'objet de différentes publications internationales dans diverses conférences (IJCAI, CP, CPAIOR), journaux (ANOR, Constraints), mais aussi l'écriture d'un livre (Wiley).

1.2. Plan du manuscrit

Tout d'abord, il est bon de rappeler que ce document est une synthèse d'une partie de mes travaux depuis 2004, la plupart des détails techniques et preuves ont été omis à la faveur d'une vue globale de la cohérence de ces derniers. Le lecteur averti pourra se référer, si nécessaire, aux publications et manuscrits de thèses afin d'y trouver les détails nécessaires à une compréhension plus fine.

La première partie de ce manuscrit, composée de trois chapitres, abordera la structure principale d'un système de programmation par contraintes moderne : les stratégies de propagation et de recherche. Les contributions sont issues de la thèse de Charles Prud'Homme et ont fait l'objet de deux publications dans le journal international *Constraints*. Dans le chapitre 2 nous commencerons par synthétiser les concepts nécessaires à une bonne compréhension de ce document, la plupart du temps nous nous référerons à l'état de l'art pour guider le lecteur vers les références pertinentes. Dans le chapitre 3, nous présenterons une première contribution liée à la définition et à la mise en œuvre d'un langage dédié pour la déclaration de politiques de propagation, chaînons aujourd'hui manquant pour permettre à un utilisateur expérimenté de configurer parfaitement le comportement du mécanisme de résolution sous-jacent. Dans le chapitre 4, nous poursuivrons par un volet sur les stratégies de recherche. Nous présenterons ainsi une nouvelle manière générique de définir un voisinage pertinent dans les

stratégies de recherche à voisinage large (LNS), qui constituent aujourd'hui les méthodes les plus efficaces pour résoudre opérationnellement des problèmes d'optimisation en programmation par contraintes. Nous montrerons comment, au delà des raisonnements basés sur la propagation qui permettent de détecter des voisinages pertinents de manière systématique, l'utilisation de la notion d'explications permet de proposer une alternative performante pour calculer d'autres types de voisinages en se basant sur la mécanique de résolution de problèmes d'optimisation en programmation par contraintes.

La seconde partie de ce manuscrit, composée de trois chapitres, sera dédiée à une étude de différents aspects des contraintes globales. Ces dernières constituent une singularité dans le paradigme car elles modélisent à elles seules un sous-problème du problème modélisé. Souvent controversées, nous nous attacherons à montrer qu'elles peuvent cependant constituer un outil puissant quand elles sont utilisées à bon escient. Ainsi, dans le chapitre 5 nous commencerons par résumer une série de deux publications (CPAIOR et CP) liées à une abstraction des contraintes de comptage au travers d'une unique contrainte globale que nous qualifierons de méta-contrainte. En effet, si les contraintes globales ont un intérêt incontestable pour traiter efficacement le filtrage d'un sous-problème fortement combinatoire, elles sont pas essence très (trop ?) spécifiques dans leur définition, ce qui peut rapidement pousser à les considérer comme un aveu d'impuissance à rester dans la philosophie du paradigme. Ainsi, s'interroger sur la mutualisation et l'abstraction des propriétés de filtrages liées à certaines contraintes globales est un travail nécessaire et bénéfique pour maintenir un nombre cohérent de contraintes globales [BEL 10a]. Dans le chapitre 6, nous nous intéressons aux objets fortement combinatoires que sont les graphes et de leur intégration dans les algorithmes de filtrages en programmation par contraintes. Précisément, nous présenterons une évolution significative d'une contrainte de partitionnement de graphe par des arbres, publiée lors de la conférence CP. Au delà, de cette contrainte nous tenterons de justifier la nécessité de l'utilisation des contraintes globales pour filtrer efficacement dans les problèmes issus de la théorie des graphes. Pour terminer, dans le chapitre 7, nous résumerons un travail plus exotique dans la communauté CP. Il est né de l'observation récurrente d'un biais dans l'utilisation des contraintes globales qui montre que le filtrage associé à chacune est appliqué de manière trop systématique dans la propagation alors que l'état courant des domaines des variables peut laisser présager que l'effort algorithmique ne sera pas "rentable". Pour fournir une première

réponse à ce biais, nous avons proposé une analyse en moyenne du comportement des contraintes globales, avec pour cas d'étude initial, la contrainte `allDifferent`. L'objectif final étant d'obtenir un indicateur probabiliste sur la capacité de filtrage de la contrainte durant le processus de résolution.

Enfin, nous conclurons ce manuscrit en ouvrant la piste de travaux futurs qui peuvent aujourd'hui faire sens dans la communauté.

Chapitre 2

Généralités sur la programmation par contraintes

Sommaire

L'idée intuitive de la programmation par contraintes est de proposer une méthode de résolution de problèmes combinatoires basée sur la déclaration de contraintes (pouvant être vues comme des conditions logiques portant sur des variables) devant être satisfaites par toute solution valide du problème considéré. Ainsi, dans le cadre discret, un *problème de satisfaction de contraintes* (CSP) se défini par :

– un ensemble $\mathcal{V}$ de variables (au sens mathématique du terme) prenant leurs valeurs dans un ensemble fini d'entiers;

– un ensemble $\mathcal{D}$ de domaines finis tel que $\mathcal{D}(v) \in \mathcal{D}$ représente l'ensemble des valeurs possibles pour la variable $v \in \mathcal{V}$;

– un ensemble $\mathcal{C}$ de contraintes, portant sur des sous-ensembles des variables, qui peuvent être vues comme des relations logiques entre variables.

Une *solution* d'un CSP est une affectation des variables à une valeur satisfaisant simultanément toutes les contraintes. C'est le *système de contraintes* qui contient le mécanisme de résolution permettant d'aboutir à toute ou partie de l'ensemble des solutions satisfaisant les contraintes. L'aspect déclaratif du paradigme réside dans le fait que l'utilisateur final doit seulement déclarer l'ensemble des variables, et les domaines associés, modélisant son problème et décrire les contraintes qui les lient.

Le cœur de la programmation par contraintes, et par là nous entendons son caractère opérationnel, repose sur le couple *propagation-recherche* : la partie propagation essaie de déduire de nouvelles informations à partir de l'état courant des variables. Quant à la partie recherche, elle consiste en un parcours, généralement une exploration dite en *profondeur d'abord*, de l'espace de recherche du CSP associé. Un algorithme effectuant uniquement un parcours de cet espace de recherche énumérera toutes les assignations de variables possibles jusqu'à, soit trouver une solution valide pour le CSP, soit trouver toutes les solutions, soit enfin conclure qu'il n'existe aucune solution satisfaisant simultanément toutes les contraintes. Un tel algorithme, que l'on peut qualifier d'algorithme de recherche exhaustif, a une complexité en temps exponentielle par rapport à la donnée du problème (nombre de variables, taille des domaines, nombre de contraintes). C'est à ce niveau que le mécanisme de propagation vient *intelligemment* épauler la recherche. En effet, il permet au système de contraintes de détecter et de supprimer des parties inutiles (dites *inconsistantes*) de l'espace de recherche sans avoir à les parcourir explicitement. La réduction de l'espace de recherche est effectuée au travers de la propagation de chaque contrainte

mise en jeu dans le CSP, *via* un mécanisme appelé *filtrage*. De manière intuitive et simplifiée,[1] le filtrage consiste en l'examen de chaque variable non encore assignée (ou instanciée), en vue de supprimer de son domaine toute valeur inconsistante, c'est-à-dire, toute valeur qui si elle était assignée à cette variable violerait au moins une des contraintes mises en jeu dans le CSP. Malheureusement, détecter et supprimer pour chaque variable toutes les valeurs inconsistantes relève très souvent d'un problème NP-difficile. C'est pour cela que le couple "propagation-recherche" est généralement indissociable et qu'un des enjeux majeurs de la programmation par contraintes reste la recherche d'un juste équilibre entre *l'efficacité*, en termes du temps de calcul, et le rendement effectif en terme de filtrage. Ce rendement effectif peut être vu comme la *capacité déductive* d'un algorithme de filtrage en termes de valeurs inconsistantes détectées dans le domaine de chaque variable qu'il met en jeu. En d'autres termes, lors de la conception de contraintes, il est primordial de trouver un équilibre entre le temps de calcul de l'algorithme effectuant le filtrage de la contrainte et la quantité effective de valeurs inconsistantes détectées, ce qui permet de réduire la taille de l'espace de recherche à parcourir.

2.1. Qu'est-ce qu'une variable ?

La programmation par contraintes est un paradigme dans lequel les concepts sont décrits de manière très générale. En ce sens, on admet souvent que les éléments composant le domaine d'une variable sont des symboles et que la seule propriété nécessaire est d'assurer que les domaines prennent leurs valeurs dans un ensemble fini de symboles. Cependant, on en distingue principalement deux types : les nombres entiers et les ensembles finis de nombres entiers. Cette distinction conduit à définir de manière spécifique deux types de variables que sont les variables entières et les variables ensemblistes. Elles peuvent être définies de la manière suivante :

Définition 2.1 (Variable entière)**.** *Une* variable entière v_i *prend sa valeur dans un ensemble fini d'entiers noté* $\mathcal{D}(v_i)$*. La plus petite et la plus grande valeur de* $\mathcal{D}(v_i)$ *sont respectivement notées* $\min(v_i)$ *et* $\max(v_i)$.

Définition 2.2 (Variable ensembliste)**.** *Le domaine d'une* variable ensembliste v_s *est un ensemble d'ensembles finis d'entiers. Il est décrit par sa* borne inférieure *(ou*

1. Nous ne considérons pas ici le cadre des consistances fortes (Section 2.3).

noyau*)* $\underline{v_s}$ *et sa* borne supérieure *(ou* enveloppe*)* $\overline{v_s}$*. Toute valeur de* $\underline{v_s}$ *est contenue dans* v_s *et toute valeur de* v_s *est contenue dans* $\overline{v_s}$*. Lorsque la variable ensembliste est* fixée *alors* $\underline{v_s} = \overline{v_s}$*. Les valeurs dans* $\underline{v_s}$ *sont dites* valeurs obligatoires *et les valeurs de* $\overline{v_s} \setminus \underline{v_s}$ *sont dites* valeurs potentielles*.*

Nous ne détaillerons pas les différentes stratégies permettant d'implémenter ces deux types de variables. Le lecteur doit retenir qu'elles diffèrent simplement par la définition de ce qu'est une valeur dans le domaine. Dans le cas d'une variable entière, il s'agit d'un entier, alors que dans le cas d'une variable ensembliste, il s'agit d'un ensemble fini d'entiers. De plus, il existe aujourd'hui d'autres types de variables plus complexes, comme les *variables graphe* [PAP 02, DOO 05] que nous détaillerons plus formellement dans le chapitre 5, dédié à un algorithme de filtrage portant sur les graphes, ou comme les variables sur les intervalles, dédiées à la modélisation de problèmes de satisfaction de contraintes numériques.

2.2. Qu'est-ce qu'une contrainte ?

Une contrainte peut se résumer à une relation logique entre des variables impliquées dans un CSP. Opérationnellement, une contrainte offre au moins deux services au moteur de résolution qui l'utilise. Tout d'abord, elle permet de vérifier la cohérence du CSP de manière locale (c'est-à-dire, relativement à la propriété qu'elle maintient), cette vérification passe généralement par une condition nécessaire (parfois aussi suffisante, mais ce n'est pas un service offert par tous les outils) pour la propriété. Ensuite, elle a pour objectif de restreindre les valeurs qui peuvent être affectées à ses variables. Cette restriction est exclusivement le fruit d'un raisonnement logique. Nous entendons par là que pour toute valeur supprimée, on a la garantie qu'elle n'appartient à aucune solution satisfaisant la contrainte et par extension le CSP l'impliquant.

De manière très classique, on distingue deux types de contraintes : les contraintes en extension et en intention. Les *contraintes en extension* reposent généralement sur une représentation de la relation logique sous la forme d'un ensemble de tuples autorisés (ou réciproquement interdits). Nous ne détaillerons pas les techniques pour mettre en œuvre efficacement ce type de contraintes. Par opposition, les *contraintes en intention* exploitent une ou plusieurs propriétés mathématiques permettant d'exprimer la relation logique entre les variables impliquées. Considérons la contrainte $C : x \leq y$

telle que $\mathcal{D}(x) = \{2, \ldots, 6\}$ et $\mathcal{D}(y) = \{0, \ldots, 4\}$. Il faut que $\mathcal{D}(x)$ soit restreint à $\{2, \ldots, 4\}$ et $\mathcal{D}(y)$ à $\{2, \ldots, 4\}$. La contrainte C offrira les services suivants :

– *condition nécessaire* : si $\min(x) \leq \max(y)$ alors, il existe $\ell_x \in \mathcal{D}(x)$ et $\ell_y \in \mathcal{D}(y)$ tels que le tuple (ℓ_x, ℓ_y) satisfait la contrainte C;

– *condition suffisante* : si quelle que soit $\ell_x \in \mathcal{D}(x)$ et $\ell_y \in \mathcal{D}(y)$, le tuple (ℓ_x, ℓ_y) satisfait la contrainte C alors, $\max(x) \leq \min(y)$;

– *filtrage* : si $\min(y) < \min(x)$ alors $\min(y) \leftarrow \min(x)$, si $\max(x) > \max(y)$ alors $\max(x) \leftarrow \max(y)$.

Le lecteur aura noté que seule la condition nécessaire suffit à assurer la correction et la complétude du paradigme. En effet, comme énoncé précédemment, le filtrage lié à une contrainte est uniquement là pour améliorer les performances de l'algorithme en diminuant la taille de l'espace de recherche. Une condition suffisante a pour but, là encore, d'améliorer les performances de l'algorithme. En effet, lorsque cette dernière est vérifiée, on sait que la contrainte concernée sera valide quelles que soient les valeurs affectées aux différentes variables présentes.

Finalement, rappelons encore que le choix de la programmation par contraintes est souvent motivé par le fait qu'elle permet d'exprimer de manière transparente contraintes linéaires (par exemple $x \leq y$) et contraintes non linéaires (par exemple $x \neq y$). Attention, les contraintes non linéaires sont souvent exprimables en programmation linéaire en nombre entier, soit par un processus complexe de linéarisation, soit par une modélisation subtile. Typiquement, la contrainte $x \neq y$ pourra s'exprimer par les deux contraintes linéaires :

$$x - y < K \cdot (1 - z) \tag{2.1}$$

$$y - x < K \cdot z \tag{2.2}$$

où K est une constante entière supérieure à la plus grande borne supérieure entre x et y, et z une variable définie sur $\{0, 1\}$. On comprend que cette modélisation est loin d'être efficace puisqu'elle nécessite deux contraintes linéaires et l'introduction d'une variable supplémentaire pour exprimer la contrainte $x \neq y$. Ainsi, l'expression simple de contraintes "complexes" est un atout majeur qui permet de distinguer la

programmation par contraintes d'autres techniques de résolution de problèmes combinatoires. Cette observation se confirme d'avantage encore lorsque l'on considère le cas exotique des *contraintes globales* [BES 03]. Ces dernières permettent en effet d'exprimer la sémantique d'une relation logique au sein d'un sous-ensemble, de taille quelconque, des variables mises en jeu dans un CSP. Nous reviendrons plus en détails sur cet objet singulier dans le paradigme CSP dans la Partie II de ce manuscrit.

2.3. Qu'est-ce qu'un algorithme de propagation ?

Etant donné un CSP, nous avons vu précédemment que chaque contrainte impliquée dans ce CSP définit un algorithme de filtrage à même de détecter un ensemble de valeurs inconsistantes dans le domaine des variables du CSP. Pourtant, les contraintes du CSP partagent très souvent des variables, ce qui implique que les algorithmes de filtrage de chacune sont dépendants. C'est dans l'expression de cette dépendance que naît la notion d'algorithme de propagation.

L'idée intuitive du terme *propagation* débute avec la notion de valeurs inconsistantes dans le domaine d'une variable vis-à-vis d'une contrainte. Cette intuition a historiquement conduit à proposer la définition de consistance d'arc.[2]

Définition 2.3 (Arc-consistance généralisée [MAC 77, MAC 85])**.** *Une contrainte C, définie sur les variables entières v_1^d, ..., v_k^d, atteint l'*arc-consistance généralisée *(GAC) si et seulement si pour chaque paire (v_i^d, ℓ) telle que v_i^d est une variable entière de C, $i \in [1; k]$, et $\ell \in \mathcal{D}(v_i^d)$, il existe au moins une affectation totale des variables de C, respectant C, dans laquelle la valeur ℓ est assignée à la variable v_i^d.*

Au final, un CSP est dit globalement arc-consistant si, et seulement si, chaque contrainte atteint l'arc-consistance généralisée. On peut donc décrire l'idée intuitive du déroulement d'un algorithme de propagation par le tableau de la figure 2.1. Ce tableau décrit le mécanisme d'un algorithme de filtrage classique au travers d'un CSP composé de trois variables x, y, z, respectivement définies par les domaines $\{5, \ldots, 10\}$,

2. Dans le cadre de contraintes binaires, l'association de deux variables pour une contrainte est nommée *arc*. Sans perte de généralités, on peut aussi nommer arc l'association variable-contrainte, pour peu que l'on observe le réseau par sa matrice d'incidence.

$\{1,\ldots,8\}$ et $\{1,\ldots,6\}$, et de quatre contraintes C_1 : *Pair*(x), C_2 : $y = x - 1$, C_3 : $y \leq z$ et C_4 : $x \neq z$. La première ligne du tableau donne l'état initial (pas 0) de l'algorithme. La colonne "à propager" donne la liste des contraintes dont la cohérence est encore à vérifier (ici, on confondra la cohérence avec l'application de l'algorithme de filtrage relatif), la colonne "cohérente" donne la liste des contraintes dont la cohérence est actuellement vérifiée. Notons que tant qu'il reste des contraintes dans la colonne "à propager" l'appartenance d'une contrainte à la colonne "cohérente" n'est éventuellement que transitoire. L'algorithme de propagation s'arrête (on dit qu'il a atteint un *point fixe*) lorsque l'ensemble des contraintes "à propager" est vide. Un pas quelconque de l'algorithme consiste donc à choisir une contrainte "à propager", à appliquer son algorithme de filtrage (il s'en suit une éventuelle modification des domaines des variables), puis à placer cette contrainte dans la colonne "cohérente" et enfin, il faut replacer dans la colonne "à propager" toutes les contraintes "cohérentes" dont au moins un domaine des variables a été modifié par le filtrage de la contrainte choisie.

# pas	x	y	z	à propager	cohérente
0	$\{5,\ldots,10\}$	$\{1,\ldots,8\}$	$\{1,\ldots,6\}$	$C_1, C_2, C_3, \mathbf{C_4}$	$\emptyset$
1	-	-	-	$C_1, \mathbf{C_2}, C_3$	$\mathbf{C_4}$
2	$\{5,\ldots,\mathbf{9}\}$	$\{\mathbf{4},\ldots,8\}$	-	$C_1, \mathbf{C_3}, C_4$	$\mathbf{C_2}$
3	-	$\{4,\ldots,\mathbf{6}\}$	$\{\mathbf{4},\ldots,6\}$	$\mathbf{C_1}, C_4, C_2$	$\mathbf{C_3}$
4	$\{\mathbf{6},\mathbf{8}\}$	-	-	$\mathbf{C_4}, C_2$	$C_3, \mathbf{C_1}$
5	-	-	-	$\mathbf{C_2}$	$C_3, C_1, \mathbf{C_4}$
6	$\{\mathbf{6}\}$	$\{\mathbf{5}\}$	-	$C_3, C_1, \mathbf{C_4}$	$\mathbf{C_2}$
7	-	-	$\{4,\ldots,\mathbf{5}\}$	$\mathbf{C_3}, C_1$	$C_2, \mathbf{C_4}$
8	-	-	$\{\mathbf{5}\}$	$\mathbf{C_1}, C_4$	$C_2, \mathbf{C_3}$
9	-	-	-	$\mathbf{C_4}$	$C_2, C_3, \mathbf{C_1}$
10	$\{6\}$	$\{5\}$	$\{5\}$	$\emptyset$	$C_2, C_3, C_1, \mathbf{C_4}$

Figure 2.1. *Illustration des étapes d'un algorithme de propagation*

Dans le contexte de notre CSP, supposons que la contrainte C_4 soit choisie (en gras sur la ligne 0), l'algorithme de filtrage de C_4 ne supprime aucune valeur des domaines de x et z car leurs domaines sont cohérents vis-à-vis de la contrainte "différent". Au pas 1, supposons que la contrainte C_2 soit choisie alors, l'algorithme de filtrage de C_2 modifie les bornes inférieures et supérieures de respectivement y et x. Cette modification implique que la cohérence de la contrainte C_4 doit à nouveau être vérifiée donc C_4 sera à nouveau placée dans la colonne "à propager" au pas suivant. Quant à C_2, elle est naturellement placée dans la colonne des contraintes "cohérentes". L'algorithme continue ainsi son exécution jusqu'au point fixe, caractérisé par le fait que la liste

des contraintes “à propager” soit vide. Attention, on comprend intuitivement que si la monotonie des contraintes nous garantit de converger vers un point fixe unique, la vitesse de convergence de l'algorithme (c'est-à-dire que le nombre de pas nécessaires) peut varier suivant l'ordre dans lequel sont choisies les contraintes “à propager”. En effet, sur l'exemple précédent, propager dans l'ordre $C_1, C_2, C_3, C_2, C_4, C_1$ assure la convergence vers le point fixe en un nombre d'itérations plus faible.

Différents algorithmes fins de propagation, basés sur l'arc-consistance, ont été proposées dans l'état de l'art depuis 1977 et jusqu'au milieu des années 2000. Tout commence avec l'algorithme AC-3 [MAC 77], AC-4 [MOH 86], AC-6 [BES 94], AC-7 [BES 95, BES 99], AC-8 [CHM 98], AC-2001 [BES 05b], AC-3.2, AC-3.3 [LEC 03]. Les travaux de [RÉG 05] ont permis de proposer un algorithme d'arc consistance générique, configurable et adaptatif, appelé CAC, permettant de combiner les techniques des algorithmes d'AC existantes. D'autres niveaux de consistance, plus forts, ont également été étudiés, comme la consistance de domaine [DEB 01, BES 08b], ou plus faibles, comme la consistance aux bornes ou consistance d'intervalles, ou encore plus récemment, des travaux comme [BAL 13] qui étudient l'adaptation de la consistance au problème durant pendant la résolution. On parle d'*arc consistance généralisé* lorsque le réseau de contraintes est composé de contraintes d'arité non fixée.

Dans la suite, le chapitre 3 reposera principalement sur ces notions, en se penchant plus précisément sur la mise en œuvre de ces différents algorithmes de propagation dans le contexte des systèmes de contraintes, et particulièrement sur le cas de l'outil Choco. Au vu de la diversité et de la subtilité de ces algorithmes, nous proposons un langage déclaratif permettant de définir au niveau du modèle, le type d'algorithme de propagation à utiliser afin d'améliorer le processus de résolution global.

2.4. Qu'est-ce qu'un système de contraintes ?

Finalement, il ne nous reste plus qu'à présenter le schéma d'articulation global pour décrire entièrement ce qu'est un système de contraintes. En effet, nous avons présenté ce que sont les variables et les contraintes (briques de base), nous avons illustré brièvement ce que sont les mécanismes de propagation. Cependant, lorsque la propagation n'est pas suffisante pour conduire à une affectation complète du CSP, que faire ? En effet, toute décision arbitraire qui viendrait réduire l'espace de recherche

(c'est-à-dire, le domaine d'un ensemble de variables du CSP) conduirait à l'incomplétude de l'algorithme de résolution. Il faut donc mettre en place un mécanisme de test systématique capable de remettre en question ces choix arbitraires.

Considérons un CSP composé de quatre variables x, y, z, t, respectivement définies par les domaines $\{1, 2\}$, $\{1, 2\}$, $\{2, 3\}$ et $\{3, 4\}$ et supposons que chacune des variables prennent deux à deux des valeurs distinctes. La propagation telle que décrite précédemment ne déduit aucune valeur impossible dans le domaine des variables. Donc, pour arriver à une solution ou à énumérer l'ensemble des solutions, le système doit être capable de faire des choix arbitraires qu'il pourra aisément remettre en question. La figure 2.2 présente un arbre de recherche binaire, dit en *profondeur d'abord*. Chaque carré donne l'état courant du CSP après propagation d'un choix arbitraire, porté par l'arête entre deux carrés.

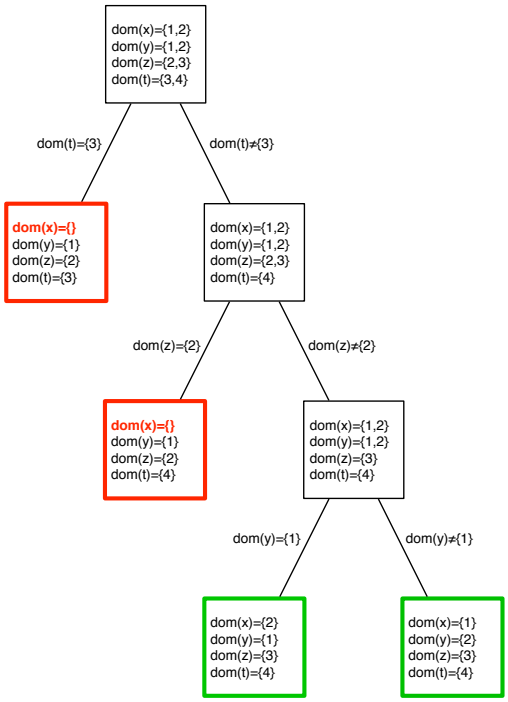


Figure 2.2. *Un arbre de recherche*

Très souvent un choix correspond à l'affectation d'une variable à une valeur de son domaine (appelé communément, une *décision*). Cependant, rien n'oblige à cela, en effet, un choix peut se généraliser à une restriction du domaine d'une ou plusieurs variables. C'est le schéma de branchement général qui garantira la complétude de l'approche. En pratique, les deux heuristiques les plus utilisées consistent à choisir d'abord (1) la variable la *plus contrainte* dans le CSP, c'est-à-dire, la variable dont le domaine est le plus restreint à cet instant ou (2) la variable la *plus contraignante* pour le CSP, c'est-à-dire celle qui est impliquée dans le plus grand nombre de contraintes. La philosophie générale de ces schémas de branchements consiste à vouloir propager un maximum d'informations dans le CSP à partir de choix arbitraires apportant un minimum de changements.

Il existe cependant, une très large variété de stratégies de recherche issues du monde de la recherche opérationnelle ou de l'intelligence artificielle. On pourra mentionner le parcours en largeur d'abord, BFS ("*Breadth First Search*") qui, bien qu'il soit largement utilisé en recherche opérationnelle pour résoudre des problèmes d'optimisation, est peu utilisé en programmation par contraintes parce qu'il sollicite beaucoup trop la mémoire. On pourra également citer les explorations basées sur la notion de *divergence*, comme LDS [HAR 95] ("*Limited Discrepancy Search*"), DDS [WAL 97] ("*Depth-bounded Discrepancy Search*") ou DBDFS [BEC 00] ("*Discrepancy-Bounded Depth First Search*"). Elles reposent toutes sur le principe qu'une stratégie de branchement adaptée fait peu d'erreurs. Donc, les chemins de décisions avec beaucoup de réfutations (s'écartant donc de la préconisation de la stratégie) doivent être considérés comme moins intéressants. Un chemin de décisions allant du du nœud racine à une feuille qui ne comporte qu'une seule décision réfutée, c.-à-d. une seule divergence, devra être testé avant les chemins avec plus de divergences.

Classiquement, les problèmes rencontrés sont souvent formulés sous la forme de *problèmes d'optimisation sous contraintes*. Dans ce cadre, les procédures de recherche complète sont souvent laborieuses, par le fait que converger rapidement vers une *bonne* solution est souvent opposé à la stratégie consistant à prouver l'optimalité d'une solution courante. C'est pour cela que la communauté s'est rapidement inspirée des méthodes de résolution à base de recherche locale afin de profiter à la fois des capacité déductive de la propagation et de l'extrême efficacité de la recherche locale. Ainsi, un problème d'optimisation sous contraintes, ou COP (pour "*Constraint Optimization*

Problem") est composé de variables, d'un domaine et de contraintes, tout comme un CSP, mais également d'une fonction de coût sur une variable caractérisant le résultat de la fonction de coût à optimiser. L'objectif est alors de trouver une solution qui maximise ou minimise la valeur affectée à la variable *objectif*. Chaque nouvelle solution S impose une *coupe* sur la variable objectif, notée C_S, bornant son domaine. Une coupe impose que la prochaine solution doit être *meilleure* que la solution courante, jusqu'à ce que la solution optimale soit atteinte.

Pour résoudre les COP, la technique de Recherche à Voisinage Large [SHA 98], qui s'inspire des techniques de recherche locale pour explorer les voisins d'une solution, a présenté de bons résultats. C'est sur cet aspect de la résolution de problèmes d'optimisation que nous reviendrons tout au long du Chapitre 4 afin de proposer une contribution originale combinant la recherche à voisinage large en programmation par contraintes [PER 03] avec les techniques à base d'explications [JUS 02, JUS 00b], afin de définir de manière générique les voisinages pertinents à explorer.

PREMIÈRE PARTIE

Flexibilité d'un système de contraintes

Les systèmes de programmation par contraintes sont encore aujourd'hui dépendants de l'utilisateur modélisant son problème (le modeleur) mais aussi de sa capacité à configurer la stratégie de résolution adaptée à son problème pour tenter de gagner en efficacité lors de la résolution. Les outils diffèrent aujourd'hui de par les contraintes proposées (et les algorithmes de filtrages associés), les stratégies de recherche et de propagation qu'ils proposent. Il n'existe pas deux outils proposant les mêmes services (ou en tous cas un même service mis en œuvre de la même manière). Plus encore, aucun outil ne peut prétendre mettre en œuvre[1] de manière efficiente (i.e., rapidement et efficacement) la plupart des avancées faites par la communauté (explications, etc.) sans concéder une part des performances.

Dans cette partie nous allons proposer une synthèse de deux contributions issues de la thèse de Charles Prud'Homme et qui ont fait l'objet de deux publications [PRU 14a, PRU 14b]. Dans le chapitre 3, Nous nous attacherons à apporter une part de déclarativité dans la configuration du moteur de propagation, ceci afin d'autoriser au modeleur un niveau d'interaction avec le moteur de propagation équivalent à celui proposé pour la définition des stratégies et heuristiques de recherche. Le langage dédié qui sera proposé aura pour but de construire un moteur de propagation à la fois simple, flexible et qui autorise la mise en œuvre transparente de toutes les stratégies de propagation actuelles et plus encore. Dans le chapitre 4, nous aborderons la notion de stratégie de recherche à base de voisinage large (LNS) pour la résolution de problèmes d'optimisation. Nous montrerons comment utiliser la notion d'explications pour calculer de manière générique et dynamique des voisinages dans les méthodes de type LNS, et ce, sans pour autant payer le surcoût de gestion lié aux explications lorsque l'utilisateur ne souhaite pas les utiliser. Nous évaluerons comment cette approche combinant recherche locale et explications permet de résoudre efficacement des problèmes d'optimisation et se compose naturellement avec les méthodes de la littérature.

1. Il est aujourd'hui impossible au développeur d'un système de contraintes d'intégrer rapidement la totalité des avancées de la communauté.

Chapitre 3

Configuration de la propagation dans un système de contraintes

Sommaire

Le moteur de propagation est au cœur des systèmes de contraintes comme le moteur d'inférence l'est pour les systèmes de programmation logique. Il constitue un point d'intérêt récurrent dans la communauté [MAC 77, SCH 08] bien qu'il ne constitue pas la partie complexe des systèmes : en effet ces mécanismes reposent, comme nous l'avons vu au chapitre 2, sur des algorithmes polynomiaux en la taille du problème modélisé (variables, domaines, contraintes). Aujourd'hui, différents algorithmes existent [BES 06], en pratique, les versions mises en œuvre dans les systèmes sont principalement des variantes des algorithmes d'arc-consistance [MAC 77, MCG 79, Van 92]. Cette notion de *variantes* pour un algorithme de propagation est importante quand il s'agit de comprendre les différences de comportements entre deux systèmes. Une différence principale est par exemple liée à la notion d'*orientation* de l'algorithme de propagation. L'orientation définit le type d'information à mémoriser (en termes de structures de données) pour propager correctement les effets d'une réduction du domaine d'une variable au travers du réseau de contraintes. Aujourd'hui, les systèmes sont principalement basés sur deux types d'orientations : les systèmes centrés sur les variables et ceux centrés sur les contraintes. Pour exemple, IBM CPO [IBM 13], Choco [TEA 13a], Minion [TEA 13d] et OR-tools [TEA 13e] raisonnent sur les modification de l'état des variables, alors que Gecode [TEA 13b], SICStus Prolog [CAR 13] et JaCoP [TEA 13c] se basent sur les contraintes qui doivent être à nouveau vérifiées.

Concevoir un moteur de propagation, et plus généralement un système de programmation par contraintes, est avant tout une question de choix et de compromis avec l'objectif de combiner flexibilité (à la fois pour l'utilisateur mais aussi en vue des évolutions futures) et efficacité (les performances restant au cœur des préoccupations des utilisateurs). Obtenir un moteur de propagation correct, efficace et adapté à l'interaction avec les autres parties de l'outil (algorithmes de recherche, gestion mémoire, interface utilisateur) est une tâche tellement délicate que le développeur de l'outil est rarement enclin à la remettre en question. Pire, le côté déclaratif cher au paradigme de la programmation par contrainte est ici totalement effacé tant l'ouverture du moteur de propagation au niveau de l'utilisateur nécessite de reconsidérer le compromis flexibilité - efficacité. En effet, "ouvrir"ce dernier sans poser de garde-fous pour maintenir à minima la correction de l'algorithme est un choix aujourd'hui difficilement acceptable, et, bien entendu, assurer des garanties à l'utilisateur ne peut se

faire sans surcoût algorithmique. Par conséquent, les moteurs de propagation sont aujourd'hui inaccessibles au niveau de l'utilisateur - modeleur. Cependant, au vu de la complexité de la tâche, il faut s'interroger sur l'utilité d'ouvrir (même partiellement) le moteur de propagation au niveau de la modélisation, comme c'est aujourd'hui le cas pour les stratégies et heuristiques de recherche. La base de cette réflexion repose sur l'identification de la cible : ici il s'agit clairement du modeleur ayant une connaissance avancée du paradigme et des algorithmes de consistance. L'objectif est de proposer une service supplémentaire au niveau de la modélisation permettant de prototyper et comparer différentes stratégies de propagation sans pour autant avoir à connaître la mécanique interne du système sous-jacent, exactement comme proposé actuellement par les langages de modélisation pour les stratégies de recherches. L'espérance de gain reste cependant modérée puisqu'il s'agit d'optimiser le comportement d'un algorithme connu comme polynomial. Au delà cet aspect "utilité"pour la communauté, un objectif plus personnel dans la mise en place de cette démarche restait la remise à plat de l'architecture du l'outil Choco : Quelle meilleure manière de s'approprier la mécanique interne d'un outil que de repenser entièrement son moteur de propagation ?

Ainsi, dans ce chapitre, nous allons résumer le travail effectué dans la thèse de Charles Prud'Homme, en co-encadrement avec Rémi Douence, Narendra Jussien et moi-même [PRU 14a], qui présente un langage dédié à la configuration du moteur de propagation au niveau de la couche de modélisation. Les contributions sont de trois ordres. Tout d'abord, nous présenterons le langage dédié (DSL) et les services qu'il propose. Nous montrerons qu'il permet d'exprimer une très large variétés de stratégie de propagation. Ensuite, nous nous attacherons à résumer ici les propriétés qui font la pertinence de ce langage et qui assureront au modeleur de produire un algorithme correct et complet en termes de propagation. Enfin, nous montrerons comment nous avons mis en place ce langage dans une version étendue du langage de modélisation MiniZinc [G12 07]. Nous illustrerons rapidement un exemple d'utilisation avec Choco comme outil cible.

3.1. Un langage dédié pour décrire la propagation

Les langages dédiés (ou *Domain Specific Languages* - DSL- en anglais) [DEU 00] ont été imaginés afin de permettre la définition ou l'extension de langages existant afin

de traiter de façon spécifique des singularités dont les moteurs de propagation font parties. En effet, dans la course à la déclarativité, chère au paradigme de la programmation par contraintes, les concepteurs de systèmes ont rapidement souhaité “ouvrir” leur système au delà de la simple déclaration d'un modèle. Jusqu'ici seules les stratégies de recherche ont été adressées de manière pertinente [IBM 13, SCH 12]. Nous proposons ici une première contribution qui autorise une ouverture de l'algorithme de propagation au modeleur. Nous commencerons tout d'abord par lister les composants essentiels d'un systèmes de programmation par contraintes qui pourra pleinement bénéficier de notre langage. Ensuite, nous présenterons une synthèse du langage, ses bonnes propriétés, et les garanties que nous sommes en mesure d'assurer.

3.1.1. *Pré-requis*

Nous commençons ici par lister trois ingrédients essentiels pour assurer l'ouverture d'un moteur de propagation par un DSL. Ces trois schémas d'implémentations sont supposés être existants, sous une forme ou une autre, dans tout système de contraintes moderne. Ils apportent, chacun à leur façon, de la flexibilité et de l'efficacité dans la travail du moteur de propagation :

– Propagateurs priorités (ou *Staged* propagators en anglais) [SCH 08] : L'objectif est d'être capable d'associer à chaque propagateur une priorité afin de déterminer quel est le prochain à exécuter. L'approche proposée dans l'état de l'art se base principalement sur l'arité du propagateur mais aussi sur sa complexité au pire cas ;

– Regroupement de propagateurs (ou Grouped propagators en anglais) [LAG 09] : Plutôt que de voir les propagateurs comme des singularités, nous acceptons ici le schéma d'implementation *composite* où un ensemble de propagateurs peut être lui même vu comme un propagateur ;

– Accès libre aux propriétés des variables et des propagateurs : Typiquement, nous supposerons que le système est équipé de primitives permettant de récupérer la cardinalité d'une variable, l'arité d'un propagateur ou bien sa priorité.

Les propriétés relatives aux variables et aux propagateurs ont deux manières d'être évaluées : statiquement ou dynamiquement. La dernière se paie bien sur au prix d'une complexité pratique plus élevée de part les structures de données nécessaires [BOU 04].

Dans un souci de flexibilité et de précision, nous allons supposer que l'accès aux arcs du réseau de contraintes associé au CSP est explicite. Ceci peut être rapidement mis en place à l'aide de *watched literals* [GEN 06] ou d'*advisors* [LAG 07]. Bien sur, cette représentation explicite des arcs du réseau a coût mémoire, puisque de l'ordre de $\mathcal{O}(|X| \times |\mathcal{P}|)$ création d'objets sont requises, chacun d'eux maintenant un pointeur vers la variable concernée et un pointeur vers le propagateur impliqué. Cependant, l'introduction explicite des arcs au cœur du système apporte un dose de flexibilité, en autorisant l'accès à des propriétés combinant variables et propagateurs.

3.1.2. *Description*

La langage que nous proposons se décompose en deux parties (Figure 3.1) : tout d'abord une manière naturelle de regrouper les paires variables-propagateurs au sein d'ensembles homogènes d'éléments qui peuvent être propagés ; ensuite, une manière déclarative de définir la politique de propagation de chaque ensemble précédemment défini mais aussi la manière de gérer la propagation de ces ensembles entre eux. Ainsi, notre langage permet de découpler au maximum le côté déclaratif permettant de décrire les groupes d'éléments à propager, du caractère opérationnel, c.-à-d., les algorithmes de propagations qui seront utilisés pour atteindre le point fixe recherché dans la définition de chaque groupe et des groupes entre eux. Concrètement, les éléments

Figure 3.1. *Point d'entrée du langage*
$\langle propagation\_engine \rangle$ ::= $\langle group\_decl \rangle + \langle structure\_decl \rangle$ **;**

manipulés sont des *arcs*, c.-à-d. les paires variables-propagateurs définissant la structure du réseau de contraintes. La syntaxe est présentée en BNF,[1] avec les conventions suivantes : un terme en police typewriter `tt` indique un symbole terminal ; un terme en italic encadré de chevrons $\langle it \rangle$ indique un symbole non terminal ; des crochets [e] définissent une option ; des doubles crochets [[a-z]] définissent un intervalle ; le symbole plus $e+$ définit une séquence de une ou plusieurs répétitions de e ; le symbole

1. La description des règles syntaxiques du langage se fera sous la forme de Backus-Naur (abrégée en BNF). Une telle notation distingue les méta-symboles, les terminaux et les non-terminaux. Les symboles non-terminaux sont les noms des catégories que l'on définit, tandis que les terminaux sont des symboles du langage décrit. Ces symboles, ainsi que les règles de production constituent la grammaire formelle du langage.

étoile $e\star$ définit une séquence de zéro ou plus répétitions de e ; l'ellipse $e,\ldots$ définit une séquence non vide de e séparés par des virgules ; l'alternance $a|b$ indique des alternatives.

La déclaration des ensembles d'éléments repose sur la définition de l'état de l'art de *groupes de propagateurs* données dans [LAG 09]. La notion de groupe proposée dans l'état de l'art a un objectif simple mais puissant : contrôler un ensemble de propagateurs en définissant, pour le groupe, une politique interne de planification et d'exécution. Ici, nous substituons aux propagateurs la notion d'arcs afin de se placer à un niveau de granularité plus fin. Constituer des groupes d'arcs doit se faire dans le respect des règles d'affectation (décrites dans la Figure 3.2). Bien que le langage soit constitué d'arcs, l'utilisateur n'a accès qu'aux variables et aux contraintes lors de la modélisation pour définir les groupes. Les variables et contraintes, et leurs propriétés, peuvent ainsi être directement référencées dans cette partie du langage. Le

Figure 3.2. *Définition des groupes d'arcs*

⟨*group_decl*⟩ ::= ⟨*id*⟩ : ⟨*predicates*⟩ ;
⟨*id*⟩ ::= [[a-zA-Z]][[a-zA-Z0-9]]⋆
⟨*predicates*⟩ ::= ⟨*predicate*⟩ | `true`
| (⟨*predicates*⟩ ((`&&` | `||`) ⟨*predicates*⟩)+)
| `!` ⟨*predicates*⟩
⟨*predicate*⟩ ::= `in(`⟨*var_id*⟩ | ⟨*cstr_id*⟩`)`
| ⟨*attribute*⟩ ⟨*op*⟩ (⟨*int_const*⟩ | `"string"`)
⟨*op*⟩ ::= `==` | `!=` | `>` | `>=` | `<` | `<=`
⟨*attribute*⟩ ::= `var[.(name|card)]`
| `cstr[.(name|arity)]`
| `prop[.(arity|priority|prioDyn)]`
⟨*int_const*⟩ ::= [+−][[0-9]][[0-9]]⋆
⟨*var_id*⟩ ::= -⋆⟨*id*⟩
⟨*cstr_id*⟩ ::= -⋆⟨*id*⟩

lecteur intéressé pourra se référer à nos travaux [PRU 14a] pour les détails relatifs à la lecture du langage. Cependant, il faut noter que la définition des groupes peut être dépendante du modèle, lorsque des variables ou des contraintes d'un problème sont référencées explicitement grâce à leurs noms. Mais elle peut également être totalement indépendante lorsque les groupes sont décrits grâce à des propriétés. Le but de ⟨*group_decl*⟩ étant de construire des sous-ensembles d'arcs, cette partie du langage est évaluée de manière impérative, c.-à-d. chaque évaluation de prédicat peut déplacer

un ou plusieurs arcs vers le groupe défini. Cependant, un arc peut répondre vrai à des prédicats de différents groupes. Pour empêcher d'avoir à discriminer les prédicats entre eux, cette partie du langage est évaluée de haut en bas et de droite à gauche. Ainsi, un arc qui répond vrai à un prédicat devient indisponible pour les prédicats suivants.

La seconde partie du langage (Figure 3.3) porte sur la description, basée sur les groupes, des structures de données. Le but ici est double : indiquer "où" placer un arc lorsqu'il doit être planifié et "comment" sélectionner le prochain qui doit être exécuté. De la déclaration structurelle résulte un arbre syntaxique hiérarchisé, basé sur une collection ⟨*coll*⟩ qui servira à la création du moteur de propagation. Une ⟨*coll*⟩

Figure 3.3. *Description de la structure de données de propagation*

⟨*structure*⟩ ::= ⟨*struct_ext*⟩ | ⟨*struct_int*⟩
⟨*struct_ext*⟩ ::= ⟨*coll*⟩ `of {`⟨*elt*⟩`,...}` [`key` ⟨*comb_attr*⟩]
⟨*struct_int*⟩ ::= ⟨*id*⟩ `as` ⟨*coll*⟩ `of {`⟨*many*⟩`}` [`key` ⟨*comb_attr*⟩]
⟨*elt*⟩ ::= ⟨*structure*⟩
| ⟨*id*⟩ [`key` ⟨*attribute*⟩]
⟨*many*⟩ ::= `each` ⟨*attribute*⟩ `as` ⟨*coll*⟩ [`of {`⟨*many*⟩`}`]
[`key` ⟨*comb_attr*⟩]
⟨*coll*⟩ ::= `queue(`⟨*qiter*⟩`)`
| [`rev`] `list(`⟨*liter*⟩`)`
| [`max`] `heap(`⟨*qiter*⟩`)`
⟨*qiter*⟩ ::= `one` | `wone`
⟨*liter*⟩ ::= ⟨*qiter*⟩ | `for` | `wfor`
⟨*comb_attr*⟩ ::= (⟨*attr_op*⟩`.`)⋆⟨*ext_attr*⟩
⟨*ext_attr*⟩ ::= ⟨*attribute*⟩ | `size`
⟨*attr_op*⟩ ::= `any` | `min` | `max` | `sum`

est définie par une structure de données abstraite et elle peut être elle-même composée d'autres structures de données abstraite et d'arcs. Il n'y a aucune garantie qu'une ⟨*coll*⟩ soit exclusivement composée d'arcs. Une structure de données définit où les éléments vont être planifiés et comment ils seront sélectionnés pour une exécution. Il en existe trois types : une queue, une liste et un tas. Structurellement, chaque type de structure de données indique où placer un élément qui doit être planifié. Il faut également définir la manière dont les ⟨*coll*⟩ vont être parcourus. Parcourir un arc revient simplement à executer son propagateur suite à la modification de sa variable. Les parcours sont décrits en associant un itérateur à une ⟨*coll*⟩. Les itérateurs de bases `one` et `wone` sont définis pour toute ⟨*coll*⟩. L'itérateur `one` parcours un seul élément d'une

⟨*coll*⟩, en respectant l'ordre structurel induit par la structure de données abstraite. L'itérateur `wone`, version courte pour "while one", appelle `one` jusqu'à ce qu'il n'y ait plus d'éléments planifiés dans la ⟨*coll*⟩. Une liste définit deux itérateurs supplémentaires : `for` and `wfor` que nous ne détaillerons pas ici. Une ⟨*struct_ext*⟩ définit une collection ⟨*coll*⟩ composée d'une liste d'éléments ⟨*elt*⟩. Un ⟨*elt*⟩ peut faire référence soit à l'identifiant d'un groupe et une instruction d'ordre (⟨*id*⟩ [`key` ⟨*attribute*⟩]), soit une autre ⟨*structure*⟩. Une ⟨*struct_int*⟩ définit une structure régulière basée sur un quantifier unique `each-as`. Une ⟨*struct_int*⟩ permet d'exprimer, de manière compacte, des structures imbriquées, là où l'utilisation de ⟨*struct_ext*⟩ impliquerait une expression verbeuse et sujette à l'introduction d'erreurs. ⟨*struct_int*⟩ indique que chacun des arcs d'un groupe doit être affecté, sous certaines conditions données par un attribut, à une collection. Elle prend en entrée un ensemble d'arcs défini par un identifiant de groupe et elle génère autant de ⟨*coll*⟩ que requis par l'instruction ⟨*many*⟩, potentiellement imbriquées les uns dans les autres. L'instruction ⟨*many*⟩ implique une itération sur les valeurs de l'⟨*attribute*⟩ et affecte à la même ⟨*coll*⟩ les éléments avec la même valeur pour l'⟨*attribute*⟩.

3.1.3. *Propriétés et garanties*

L'objectif de ce langage est de venir avec un certain nombre de propriétés qui assurent à l'utilisateur différentes garanties quant à son utilisation. Ainsi, parmi l'ensemble des propriété et garanties listées dans l'article [PRU 14a], je retiendrais plus particulièrement les quatre propriétés suivantes qui, de mon point de vue, font la pertinence de notre approche :

– Garantie de couverture : tous les arcs d'un modèle sont représentés dans le moteur de propagation produit. L'expressivité de notre langage autorise la déclaration de moteurs de propagation incomplets, dans lequel un ou plusieurs arcs peuvent être absents. Dans ce cas, une partie du modèle sera inconnue du moteur de propagation produit, et donc, ignorée lors de la propagation. Ce défaut de couverture est détectable lors de l'interprétation de la description, en maintenant simplement le compteur d'arcs affectés à un groupe, et celui des groupes effectivement utilisés.

– Unicité de la propagation : lors de l'évaluation de l'expression d'un moteur de propagation, un arc peut satisfaire plusieurs prédicats et donc être affectés à plusieurs

groupes. L'évaluation haut-bas gauche-droite du langage garantit qu'un arc n'est affecté qu'à un seul groupe, le premier dont il satisfait le prédicat. Ainsi, un arc ne peut être planifié qu'une seule fois par événement.

– Conformité : Les arcs associés à une paire propagateur-variable inexistante dans le modèle ne sont pas représentables dans le moteur de propagation. Grâce à cette propriété, aucun arc sans signification, au sens du modèle, ne peut ainsi être planifié et donc propagé pendant la phase de propagation.

– Complétude : elle est assurée par les itérateurs complets, tels que `wone` et `wfor`, associés à la collection top-niveau. Ainsi, quelques soient les itérateurs associés aux collections filles composant le moteur de propagation, la collection mère garantit qu'il ne reste aucun élément planifié à la fin de son parcours, c.-à-d. quand la phase de propagation est terminée, et que la phase de recherche va être appliquée. Sélectionner un itérateur tel que `one` ou `for` peut supprimer cette garantie de complétude.

3.2. Utilisation pratique

Nous proposons ici de montrer comment le langage se manipule en pratique et comment en quelques lignes il permet : (1) de décrire les moteurs de propagations existants dans l'état de l'art, (2) de prototyper rapidement des moteurs de propagation dédiés à un problème particulier.

3.2.1. *Mise en œuvre*

Nous proposons ici un résumé des choix de mise en œuvre effectué afin de rendre notre langage utilisable dans un système de programmation par contrainte. Extending a modeling language is a natural way to implement our DSL: it provides access to model objects, such as variables and constraints, but maintains solver independence.

ANTLR l'interprétation du langage. Nous avons sélectionné ANTLR [PAR 89] pour interpréter la grammaire définie par notre langage. ANTLR, pour "*ANother Tool for Language Recognition*", est un framework libre de construction de compilateurs utilisant une analyse LL(*). ANTLR sépare les trois étapes d'interprétation; l'analyseur syntaxique (le *lexeur*), la reconnaissance des structures de phrases (le *parseur*) et la traduction en instructions du solveur (l'*arbre syntaxique abstrait*). Une

telle séparation permet de n'avoir à adapter que l'arbre syntaxique abstrait au solveur cible, le lexeur et le parseur n'ayant pas à être modifiés.

MiniZinc pour la mise en œuvre du langage. Nous avons choisi d'étendre le langage de modélisation MiniZinc [G12 07]. Il s'agit d'un langage de modélisation de problèmes de satisfaction de contraintes simple mais expressif. Il est supporté par un nombre important de solveurs modernes. En pratique, il est nécessaire d'ajouter la description du moteur de propagation directement dans le modèle MiniZinc. Cela se fait à l'aide de deux annotations créées à cet effet. La première sert à référencer les contraintes du modèle (Figure 3.4, Ligne 2). La seconde sert à déclarer le moteur de propagation (Figure 3.4, Ligne 3). Même si MiniZinc permet la définition d'annotations plus complexes, telles que celles utilisées pour déclarer les stratégies de recherche, les annotations présentées reposent uniquement sur des chaînes de caractères. Ainsi, on préserve l'indépendance de notre langage vis-à-vis du langage de modélisation.

Figure 3.4. *Séquence magique en MiniZinc, étendu avec notre langage.*

```
1:  include "globals.mzn";
2:  annotation engine(string:  s);
3:  annotation name(string:  s);
4:  int:  n;
5:  array [0..n - 1] of var 0..n:  x;
6:  constraint count(x,0,x[0])::name("c0");
7:  constraint forall (i in 1..n - 1) (count(x, i, x[i]));
8:  solve
9:  ::  engine("
10: G1:  in(\"c0\");
11: All:  true;
12: list(wone) of {queue(wone) of{G1},
13:      All as queue(wone) of {each cstr as list(wfor)}};
14: ")
15: satisfy;
```

FlatZinc et le cas de la reformulation de contraintes. La distribution de MiniZinc inclut un langage bas niveau destiné à être interprété par les systèmes, nommé FlatZinc. FlatZinc est le langage cible dans lequel sont traduits les modèles MiniZinc. Lors de la traduction, les annotations sont automatiquement transférées au modèle FlatZinc, incluant donc les annotations précédemment introduites. Toutefois, la traduction de

MiniZinc vers FlatZinc peut induire une décomposition des contraintes définies basée sur les contraintes primitives de FlatZinc. Par exemple, en cas d'absence de la contrainte ALLDIFFERENT dans un système, la contrainte sera reformulée en une clique de contraintes d'inégalités au niveau de la couche FlatZinc. La prise en compte de ce type de reformulation automatique est un vrai challenge que nous ne sommes, aujourd'hui, pas à même remplir. En effet, toute la sémantique associée à la contrainte globale utilisée au niveau du modèle MiniZinc peut être définitivement perdue lors de la reformulation sur la couche FlatZinc, entrainant aussi une possible incohérence dans la stratégie de propagation définie au niveau MiniZinc.

3.2.2. *Description des moteurs de propagation classiques*

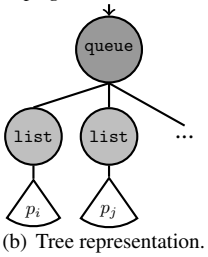
Nous nous attachons dans cette section à proposer les trois moteurs de propagations que l'on peut retrouver classiquement dans les systèmes de programmation par contraintes. Les trois moteurs exprimés par le DSL sont respectivement notés `prop`, `var` and `7qd` en référence à un raisonnement sur les propagateurs, les variables et les propagateurs avec priorités dynamiques. Le langage proposé garantit que l'expression des ces trois moteurs génère exactement le même comportement que la version native de chacun d'eux.

Tout d'abord, nous présentons comment déclarer `prop`, un moteur de propagation orienté propagateurs, à l'aide du langage (Figure 3.5). La structure prend tous les arcs du problème en entrée. Puis, les arcs sont regroupés par propagateurs dans des listes. Chaque liste définit une traversée incomplète (`for`) mais la file haut niveau assure la complétude de la propagation (`wone`). Ensuite, nous présentons la déclaration de `var`,

Figure 3.5. *Un moteur orienté propagateur.*

```
1: All:true;
2: All as queue(wone) of {
3:     each prop as list(for)
4: }
```

(a) Declaration.



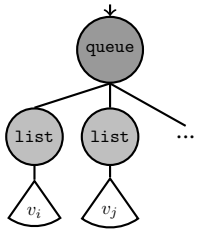
(b) Tree representation.

un moteur de propagation orienté variables, à l'aide du langage (Figure 3.6). Il est très proche de la description précédente, seulement, les arcs sont regroupés par variables. Enfin, nous présentons comment déclarer `7qd` à l'aide de notre langage, un moteur

Figure 3.6. *Un moteur orienté variables.*

```
1: All:true;
2: All as queue(wone) of {
3:     each var as list(for)
4: }
```

(a) Declaration.



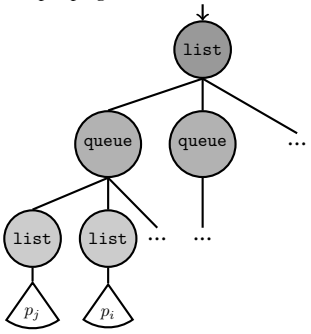
(b) Tree representation.

orienté propagateur, composé de sept files et évaluant la priorité des propagateurs dynamiquement (Figure 3.7). Comme les précédents, il prend tous les arcs du problème en entrée. Ces arcs sont répartis dans les différentes files en fonction de la priorité dynamique de leurs propagateurs. Pour chaque priorité disponible, les arcs sont regroupés autours des propagateurs dans les listes. Comme l'évaluation des priorités est dynamique, le choix de la file dans laquelle mettre une liste est fait grâce à l'évaluation du propagateur d'un arc de la liste. La liste de haut niveau garantit à la fois la complétude de la propagation (`wone`) et que les événements associés aux propagateurs de faible priorité sont traités avant ceux de plus forte priorité.

Figure 3.7. *Un moteur basé sur la priorité dynamique des propagateurs.*

```
1: All: true;
2: All as list(wone) of {
3:     each prop.prioDyn as queue(one) of {
4:         each prop as list(for)
5:     }key any.any.prop.prioDyn
6: }
```

(a) Declaration.



(b) Schéma associé.

3.2.3. *Cas d'étude*

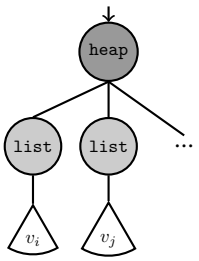
L'objet de cette section est de présenter un cas d'usage de notre langage. Nous allons nous placer dans le cas où nous souhaitons étudier l'opportunité de designer un moteur de propagation spécifique à un problème donné et d'en évaluer la pertinence face aux approches classiques de l'état de l'art. Pour cela, nous prenons comme exemple le problème de la `règle de Golomb`.[2] Le choix s'est porté sur ce problème car la structure du réseau de contraintes de ce dernier est assez singulière. En effet, Boussemart *et al.* [BOU 04] ont montré qu'orienter la propagation sur les variables de plus petite cardinalité est une très bonne stratégie de résolution.

En plus des trois stratégies classiques de propagation (précédemment introduites), nous avons retenu trois autres politiques de propagations : deux basées sur une structure de tas (`heap-var` et `heap-var-prio`) et une troisième une imbrication de deux collections (`2-coll`). La Figure 3.8 représente la première structure basée sur un tas : `heap-var`. Elle est directement dérivée de [BOU 04]. Les arcs du réseau sont

Figure 3.8. *Moteur de propagation basé sur un tas.*

```
1: All: true;
2: All as heap(wone) of {
3:     each var as list(for) key any.var.card
4: };
```

(a) Declaration.



(b) Schéma associé

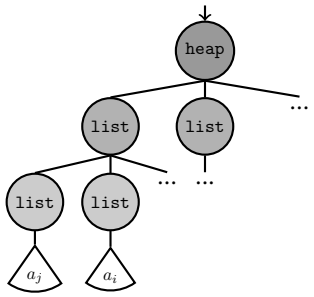
stockés dans des listes, elles-mêmes triées dans un tas. Le tas est organisé selon le critère de cardinalité des variables. La figure 3.9 présente une variante assez naturelle de la stratégie précédente. La politique `heap-var-prio` trie les arcs du réseau selon leur priorité. Par priorité nous entendons, qu'elle exécute en premier, pour une variable donnée, les propagateurs de plus faible coût au sens de la complexité. Enfin, la

2. On considère un ensemble de m ientiers $0 = a_1 < a_2 < \cdots < a_m$, représentant les m marques de la règle, telles que on pose $\frac{m(m-1)}{2}$ contraintes de différences sur $a_j - a_i, 1 \leq i < j \leq m$. Une telle règle contiendra m marques et sera de longueur a_m. L'objectif est de trouver une règle de longueur minimale.

Figure 3.9. *Variation autour du moteur de la Figure 3.8.*

```
1: All: true;
2: All as heap(wone) of {
3:     each var as list(for) of {
4:          each prop.priority as list(for)
6:          key any.prop.priority
5: } key any.any.var.card};
```

(a) Declaration.



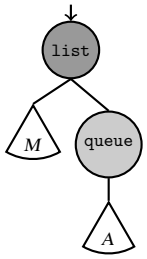
(b) Schéma associé

Figure 3.10 présente la structure `2-coll`. Elle est basée sur la composition de deux collections au sein d'une liste. La première collection gère l'ensemble des arcs `M` qui sont associés aux variables de type a. Ils sont triés dans une liste en ordre croissant sur leur nom : les évènements intervenants sur la variable a_i passent avant ceux de la variable a_{i+1}. La gestion de cette collection par le mot clé `wfor` assure de balayer itérativement la collection jusqu'à l'avoir épuisé, c.-à-d. avoir atteint un point fixe local. Alors, la second collection, gérée par une queue contenant les arcs restant est à son tour épuisée.

Figure 3.10. *Moteur basé sur une double collection ordonnée.*

```
1: M: in(mark);
2: A: true;
3: list(wfor) of {
4:   list(wfor) of {M key var.name},
5:   queue(wone) of {A}
6: };
```

(a) Declaration.



(b) Tree representation

Nous venons de montrer qu'il était possible de modéliser en quelques lignes le comportement de six moteurs de propagation distincts, dont certains seraient relativement subtils à concevoir directement au cœur d'un système de contraintes. En terme de performances, le Tableau 3.1 synthétise les résultats obtenus pour une règle de

10 marques et une autre de 11. Il est difficile d'établir une relation directe entre le nombre de propagations et le temps de résolution. Le temps de réponse repose sur un subtil équilibre entre la propagation des contraintes légères et la contrainte globale `AllDifferent`, dont le coût de propagation est important mais qui est également moins souvent propagée (puisque de priorité plus faible que les autres propagateurs du modèle). Une sélection intelligente du prochain arc à propager est donc payante. Les résultats observés sont similaires à ceux de [BOU 04]. Ainsi, la sélection de la variable de plus petite cardinalité permet de résoudre plus efficacement le problème. Enfin, le moteur `2-coll` est le plus efficace. Son comportement est comparable à celui

Moteur	$m = 10$		$m = 11$	
	`temps` (sec.)	`pex` (10^6)	`temps` (sec.)	`pex` (10^6)
`prop`	3.83	10.085	76.70	191.894
`var`	2.99	10.305	55.50	192.034
`7qd`	3.19	10.860	60.67	200.108
`heap-var`	2.79	10.717	53.09	196.525
`heap-var-prio`	3.31	11.242	59.84	207.973
`2-coll`	**2.69**	**9.975**	**52.48**	**185.921**

Tableau 3.1. *Evaluation des moteurs pour le problème de `Golomb ruler`.*

du `heap-var`, mais il ne nécessite aucune évaluation dynamique de critères, ce qui le rend légèrement plus rapide. Il est basé sur une liste et une file dont les opérations se font en temps constant. Le choix d'évaluer ce dernier moteur est important. En effet, la simplicité de déclaration d'une nouvelle politique en regard de sa mise en œuvre directe au sein d'un moteur de propagation, nous a permis de rapidement sélectionner les politiques prometteuses parmi un ensemble qui aurait été extrêmement laborieux en mettre en œuvre de manière ad-hoc.

Nous retiendrons donc la facilité avec laquelle le prototypage a été effectué avec le DSL. Il a été possible, en quelques lignes, de définir une heuristique de propagation spécifique, reposant sur les propriétés des variables et des contraintes, et de produire un moteur de propagation sûr, interprétable et utilisable en pratique. Cela simplifie grandement le processus d'évaluation et promeut l'étude de l'ordre de révision des contraintes au sein de CSP.

3.3. Conclusion et perspectives

Notre première motivation, dans ce chapitre, était de fournir un outil qui rende possible la configuration du moteur de propagation dans un système de contraintes. Pour cela, nous avons proposé un langage dédié à la description des moteurs de propagation. Le lecteur intéressé pourra se référer à nos travaux [PRU 14a] pour avoir une évaluation plus précise des surcoûts engendrés par la mise en œuvre et l'utilisation d'un tel langage (une implémentation dans le langage de modélisation MiniZinc a été proposée). Aujourd'hui et de manière synthétique, nous pouvons affirmer que le surcoût est très limité et que ce langage est au final un outil de prototypage très puissant. L'intérêt de la phase de prototypage est de construire et d'évaluer rapidement, mais sûrement, des moteurs de propagation. La phase d'analyse grammaticale n'est pas critique dans une démarche de prototypage, mais elle ne doit tout de même pas trop pénaliser les tests. La phase de résolution, quant à elle, est critique. À algorithme de propagation équivalent, les versions interprétées des moteurs de propagation ne peuvent pas être compétitives, en termes d'efficacité, avec les versions natives. Les expérimentations menées montrent que le coût d'utilisation est relativement faible et permet de valider notre approche en tant qu'outil de prototypage. Nous montrons que le surcoût induit par l'interprétation est acceptable, mais surtout que le classement des moteurs interprétés par rapport à leur temps d'exécution est généralement le même que le classement des moteurs natifs.

Il existe bien entendu des limites à la configuration des moteurs de propagation par notre approche. La première, la plus importante, mais également la plus problématique à mettre en œuvre, concerne la prise en compte de la reformulation des contraintes, et surtout son impact sur les moteurs de propagation produits. La restriction majeure réside dans les prédicats et attributs nécessaires à la définition de groupe de propagateurs [LAG 09]; des alternatives doivent être proposées. Nous avons introduit, en nous calquant sur le fonctionnement de MiniZinc concernant les contraintes non supportées par un système cible, un schéma de propagation type par contrainte globale reformulée. Bien entendu, un tel choix devrait être évalué pour juger de l'interaction des schémas de reformulation sur la propagation. Cependant l'impact même des reformulations sur la propagation n'a pas été véritablement mesuré, on s'intéresse souvent en premier au niveau de filtrage d'une reformulation. La seconde limite qui doit être considérée, inhérente à l'utilisation même d'un langage dédié, concerne la création de

moteurs de propagation “mal formés” dont il est important de prévenir la création. Une approche envisageable est d'établir des règles de transformations et d'optimisation des descriptions qui ne dénaturent pas la description initiale mais la rende plus compacte et donc évaluable plus efficacement. Notre langage pourrait alors faire partie intégrante des standards pour la programmation par contraintes (MiniZinc [G12 07], JSR331 [FEL]). Enfin, il faut reconnaître que notre ambition initiale, exploiter la structure d'un problème pour améliorer la propagation, a rapidement été confrontée à une difficulté majeure: celle de trouver un problème, non pathologique, pour lequel une description fine de la propagation soit la clé du succès. Nous n'avons, cependant, pas appliqué notre approche sur des problèmes réels, ni d'ailleurs évalué l'interaction du moteur de propagation avec la stratégie de recherche.

Chapitre 4

Explications pour la recherche à base de voisinages larges

Sommaire

Nous venons de traiter dans le chapitre précédent des aspects liés à l'ouverture des moteurs de propagation dans les systèmes de contraintes ; avant de traiter dans la prochaine partie des aspects liés au filtrage, nous allons rapporter ici le travail effectué par Charles Prud'homme durant sa thèse [PRU 14b]. Ce travail prend naissance dans le constat que pour beaucoup de problèmes d'optimisation, combiner la force du raisonnement lié à la propagation avec les principes de la recherche locale est aujourd'hui la seule voie pour les résoudre efficacement. Bien entendu, les méthodes issues de la recherche locale restent par nature incomplètes mais on peut aisément accepter aujourd'hui que pour résoudre de manière pragmatique et satisfaisante un problème d'optimisation, la preuve d'optimalité en elle-même n'est pas une nécessité opérationnelle, ou que faire la preuve d'optimalisé d'un problème ne relève pas de la même ambition que celle de découvrir rapidement une "bonne" solution.

Dans le cadre de la programmation par contraintes, la technique de recherche locale la plus utilisée est la recherche à base de voisinage large (LNS) [SHA 98, PIS 10]. Cette technique repose sur un mécanisme de redémarrage de la recherche à partir d'un sous-ensemble de variables à déterminer, le tout reposant sur une recherche arborescente classique. En d'autres termes : à partir de la meilleure solution courante, on détermine un ensemble de variables que l'on souhaite laisser fixées (fragment) et l'on restaure toutes les autres à leur état initial, on lance une nouvelle recherche arborescente non exhaustive (c.-à-d. limitée à une certaine profondeur ou à un certain nombre d'échecs) et l'on fait évoluer l'ensemble des variables fixées jusqu'à améliorer la solution courante. Comme toute technique de recherche locale, elle nécessite une phase de configuration assez délicate : il faut trouver un compromis entre intensification autour du fragment courant et diversification vers d'autres voisinages (c.-à-d. déterminer de nouveaux fragments que l'on espère plus pertinents). Par chance, cette technique s'intègre de manière tout à fait naturelle dans les systèmes de contraintes [PER 03]. Malheureusement, les méthodes pour calculer un fragment pertinent sont très souvent complètement dépendante du problème (voire de l'instance) à résoudre. Durant les dix dernières années, différents travaux ont eu pour vocation de proposer des méthodes génériques de calcul du fragment. On retient aujourd'hui principalement une méthode générique exploitant les effets de la propagation, appelée "Propagation-Guided LNS" [DAN 03, PER 04], et qui a été montrée comme très compétitive sur le problème de "car sequencing".

Dans ce chapitre nous nous proposons d'utiliser la notion d'explications afin de calculer de manière pertinente et générique des voisinages pour l'algorithme de recherche LNS. Durant la dernière décennie, les technique de recherches à base d'explications ont été de plus en plus étudiées. De manière très synthétique, les *explications* représentent une trace explicite des mécanismes de propagation et de recherche, elles rendent donc possible l'identification d'un sous-ensemble de contraintes et de décisions responsables de l'état courant des domaines de variables [OHR 09, VER 05]. Elles permettent en particulier d'identifier les structures cachées dans le réseau de contraintes [CAM 06] et de les exploiter pour améliorer la recherche [JUS 02, JUS 00b]. Jusqu'ici, les stratégies à base d'explications ont souffert d'une réelle difficulté de passage à l'échelle (explosion de la consommation mémoire et du temps de calcul) ainsi qu'une difficulté de mise en œuvre propre et non intrusive dans les outils modernes. Ainsi, nous proposons ici une synthèse de l'article [PRU 14b] qui décrit une manière efficace et flexible pour calculer des voisinages (les variables à relaxer) dans une méthode LNS à partir d'explications liées à l'état de la meilleure solution courante. Précisément, nous nous intéressons à comprendre pourquoi la solution courante n'est plus adaptée au problème ou pourquoi elle n'est plus optimale au sens de l'évolution de la fonction objectif. Nous montrons comment ces informations peuvent être calculées dans une recherche avec retour-arrière standard et ce sans payer le prix des stratégies classiques basées sur les explications. Enfin, nous montrons comment ces nouvelles techniques de gestion des explications permettant de calculer des voisinages peuvent se combiner avec les stratégies existantes afin d'obtenir un meilleur compromis diversification / intensification. Finalement, une évaluation assez exhaustive viendra confirmer l'apport de cette stratégies en se basant sur une large variété de problèmes issus de la base MiniZinc.

4.1. Calcul de voisinages expliqués

Cette section résume deux manières d'exploiter la notion d'explications dans le calcul de voisinages pour la méthode LNS. Nous allons nous placer dans le contexte de variables où les domaines sont définis par des intervalles de valeurs et non un ensemble de valeurs. Cette hypothèse est assez lourde de conséquences dans les systèmes de programmation par contraintes, en particulier lorsqu'ils sont instrumentés pour calculer les explications de chaque retrait de valeur. Ainsi, nous retenons pour

cette étude le cadre favorable aux explications des domaines définis par intervalles de valeurs.

Dans une approche LNS, la meilleure solution courante sert de point de départ au calcul du prochain voisinage à considérer. Cependant, à l'exception de méthodes spécifiques au problème considéré, l'état de l'art ne présente que peu de méthodes génériques. On retiendra surtout l'approche basée sur le raisonnement par propagation proposée dans [PER 03] qui constitue aujourd'hui la seule "offre" générique crédible et opérationnelle dans l'état de l'art. Se proposer d'analyser plus finement la meilleure solution courante conduit inévitablement à se demander en quoi elle est une bonne solution et en quoi elle pourrait être améliorée. Ceci nous a conduit naturellement à tenter de faire le lien avec les méthodes de recherches basées sur les explications.

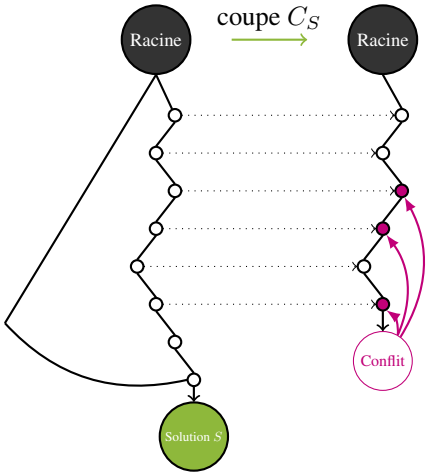
Dans la suite de cette section nous allons résumer le principe des deux méthodes à base d'explications pour le calcul de voisinage que nous avons expérimentées. Ces deux méthodes nommées respectivement `exp-cft` (Section 4.1.1) et `exp-obj` (Section 4.1.2) ont fait l'objet d'une évaluation fine sur un nombre significatif de problèmes distincts issus de la bibliothèque de problèmes MiniZinc dont une synthèse des résultats sera proposée dans la section 4.2. Pour plus d'informations sur cette section, le lecteur pourra trouver les détails algorithmiques de chaque stratégie dans [PRU 14b].

4.1.1. *Expliquer les coupes générées*

Ce premier voisinage repose sur la méthode standard pour gérer le processus d'optimisation dans les systèmes de contraintes. À chaque nouvelle solution trouvée, une contrainte interdisant de retrouver une solution de coût équivalent ou moins bon (inférieur ou égal dans le cas maximisation, supérieur ou égal dans le cas minimisation) est ajoutée au modèle, ce type de contraintes est généralement appelé *coupe*.

Dans notre démarche, c'est à ce niveau que les explications viennent proposer une manière élégante de calculer un voisinage. Considérant la solution courante et l'ensemble des décisions qui ont permis d'y aboutir (schéma à droite de la Figure 4.1), nous sommes en mesure de provoquer un conflit et en ré-appliquant les décisions menant à cette solution dans un modèle augmenté de la coupe (notée C_S dans la Figure 4.1) générée par la solution courante (schéma à gauche de la Figure 4.1). Alors,

Figure 4.1. *Schéma du voisinage* `exp-cft`.



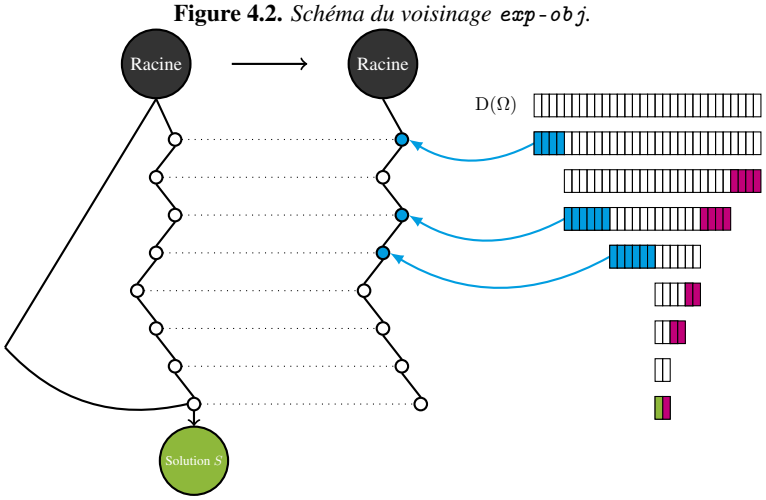
lors de la détection de l'échec à ré-appliquer les décisions, le système d'explications va détecter les décisions responsables de ce conflit (noeuds en rose sur la Figure 4.1), des quelles l'algorithme de calcul du voisinage va extraire les variables concernées. cette approche permet donc de calculer un sous-ensemble de variable qui doivent être nécessairement remise en question pour se laisser une chance de satisfaire la coupe générée par la découverte de la dernière solution.

4.1.2. *Expliquer l'évolution de la variable objectif*

Ce second voisinage est basé sur une manière moins naturelle d'exploiter le système d'explications. En effet, précédemment, nous avons provoqué un conflit explicite (reposant sur la manière de générer une coupe) pour calculer les variables à relaxer dans le prochain voisinage. Ici, nous allons raisonner sur la nature non optimale de la solution courante.

En effet, il est possible grâce aux mécanisme d'explications de comprendre la raison du retrait d'une ou plusieurs valeurs du domaines de la variable objectif. Comme précédemment, lors de la découverte d'une nouvelle solution améliorant l'objectif, nous interrogeons le système d'explications afin d'extraire l'ensemble des décisions en relation avec le retrait de valeurs inférieures, pour le cas de la minimisation, (resp. supérieures, pour le cas de la maximisation) à la valeur de la variable objectif dans la

Figure 4.2. *Schéma du voisinage* `exp-obj`.



solution courante. Au départ, les décisions sont sélectionnées pour être supprimées dans l'ordre croissant (pour le cas de la minimisation, décroissant sinon) des valeurs du domaine de la variable objectif. Lorsque toutes les décisions ont été supprimées du chemin de décisions mais qu'aucune solution n'a pu être trouvée, alors les décisions à supprimer sont choisies aléatoirement. L'exemple de la Figure 4.2 illustre schématiquement ce comportement pour le cas de la minimisation. Les noeuds en bleu illustrent les décisions qui ont entrainé une modification de la borne inférieure de la variable objectif Ω, les modifications de la borne supérieure n'ont aucun intérêt dans notre cas (marquées en rose sur le schéma du domaine). Ceux sont donc les variables extraites des décisions en bleu qui vont définir le voisinage à considérer pour `exp-obj`.

4.1.3. *Améliorations pratiques et approche retenue*

L'efficacité pratique des algorithmes de calcul de ses voisinages reposent principalement sur la capacité du système d'explications à travailler efficacement. Nous pouvons résumer les optimisations que nous avons effectuées à trois aspects :

– *Enregistrement des explications*: Les algorithmes de recherche "intelligents" accèdent à la base d'explications sur chaque conflit, ce qui n'est pas nectaire dans notre cas puisque nous souhaitons y accéder uniquement lorsqu'une nouvelle solution est

trouvée. En conséquence, il n'est pas pertinent de calculer et de mémoriser les explications au cours de la résolution. Pour limiter la consommation CPU (liée aux calculs des explications) et mémoire (liée à leur stockage), nous avons proposé une manière paresseuse et asynchrone de maintenir la base d?explications, similaire à celle décrite dans [GEN 10].

– *Gestion du retrait d'intervalles de valeurs*: Le calcul des explications passe généralement par le calcul d'une information pour chaque retrait de valeur du domaine d'une variable. Cependant, dans le cadre des variables définies par de intervalles de valeurs, il est contre-productif de gérer les retrait de valeurs un à un pour un intervalle donné, qui peut être de taille très conséquent, en particulier dans le cadre d'une variable représentant un objectif à optimiser. Nous avons donc adapté une technique issue de [JUS 98] qui permet une gestion appropriée de ce type de variables.

– *Relaxation d'un chemin de décisions* : Lorsque l'ensemble des décisions à relaxer R a été déterminé, la théorie des explications [JUS 03] nous autorise à supprimer du chemin des décisions initiales B, d'une part R, mais aussi toutes les réfutations du chemin de B qui s'expliquent par au moins une décision de R.

Pour notre évaluation finale, l'approche retenue se place dans la lignée des travaux de [PER 04]. Ainsi, nous allons combiner les deux techniques de calcul de voisinages basées sur les explications avec une troisième basée sur une génération aléatoire. Chacun des trois voisinages sera appliqué séquentiellement jusqu'à découvrir une nouvelle solution. De plus, nous acceptons sans discussion l'intégration d'une politique de "Fast Restart" limitant la phase de recherche arborescente autour d'un voisinage à trente échecs enregistrés [PER 03]. Cette approche sera notée dans la suite Explanation-Based LNS (EBLNS).

4.2. Évaluation

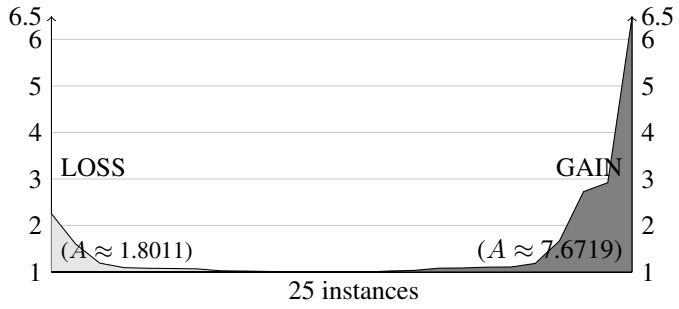
Les approches proposées ont été évaluées de manière intensives sur un ensemble de dix problèmes répartis dans 49 instances. Ces problèmes sont issus majoritairement de la base MiniZinc et plus particulièrement des Challenges 2012 et 2013. La selection des instances s'est faite sur le critère classique en optimisation qu'il était possible de trouver une première solution en moins de 15 minutes par un algorithme de backtrack classique. Nous distinguons particulièrement les problèmes de minimisation (5

problèmes) de ceux de maximisation (5 problèmes). Les approches “Propagation-Guided LNS” and “Explanation-Based LNS” ont été développées dans la version Choco-3.1.0 [TEA 13a]. Les expérimentations ont été effectuées sur un Macbook Pro avec 6-cœrus Intel Xeon cadencés à 2.93Ghz tournant sur un MacOS 10.6.8, et une version Java 1.7. Chaque exécution a été limitée sur un coeur et fait l'objet d'une limite de temps fixée à 15 minutes. Les résultats complets et détaillés sont disponibles dans [PRU 14b].

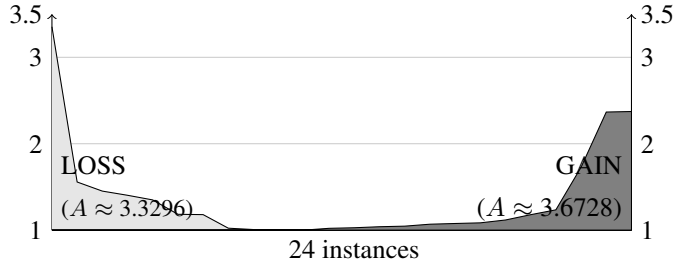
4.2.1. *Synthèse des expérimentations*

Les résultats sont globalement bien distribués entre l'approche de l'état de l'art (PGLNS) et notre proposition (EBLNS) qui combine les deux méthodes de calcul de voisinages à base d'explications. Ainsi, sur les 49 instances, 22 sont favorables à l'état de l'art et 24 notre approche. De plus, dans un tiers des cas la même valeur pour l'objectif est découverte par les deux approches dans la limite de 15 minutes.

Une analyse plus fine des résultats permet de se rendre compte plus précisément de l'apport d'une approche basée sur les explications. La Figure 4.3 propose une manière synthétique de quantifier le gain/perte de notre approche par rapport à l'état de l'art PGLNS. Cette figure se lit en observant les aires définies en gris clair et gris foncé qui correspondent respectivement à la perte et au gain de l'approche EBLNS par rapport à l'approche PGLNS. Un point dans la partie gauche du graphique (région gris clair) reporte le ratio entre PGLNS et EBLNS pour une instance mieux résolue avec PGLNS, on mesure alors la dégradation liée à l'utilisation EBLNS. Un point dans la partie droite du graphique (région bleue) reporte EBLNS et PGLNS) pour une instance mieux résolue avec EBLNS, on mesure alors l'amélioration liée à l'utilisation EBLNS. Concernant les problèmes de minimisation, les succès de chaque approche sont équitablement répartis, mais le gain lié à l'utilisation de EBLNS à la place de PGLNS est remarquable. La surface de la région en gris foncé ($A \approx 7.6719$) est clairement plus grande que celle de la région en gris clair ($A \approx 1.8011$). EBLNS améliore majoritairement les solutions, et les dégrade dans une moindre mesure. Concernant les problèmes de maximisation, le gain est moins marqué, mais légèrement en faveur de EBLNS. Les surfaces sont comparables, mais EBLNS traite mieux plus d'instances que PGLNS: la surface gris foncée ($A \approx 3.6728$) est plus large que haute.



(a) Cas de la minimisation.



(b) Cas de la maximisation.

Figure 4.3. *Comparaison du ratio entre PGLNS et EBLNS par instances.*

Un fait remarquable concerne la proportion d'instances équivalentes qui est assez faible, quel que soit le type de problèmes. Cette séparation des résultats laisse imaginer que les deux approches pourraient potentiellement bien se combiner, ce que nous allons confirmer dans la suite.

4.2.2. *Combiner les approches*

Comme nous l'avons déjà montré, une des forces des schémas LNS en programmation par contraintes sont d'être fortement modulables, dans le sens où l'on peut facilement séquencer différents algorithmes de calculs des voisinages afin de favoriser la diversification des fragments produits. De ce constat, il devait donc tout naturel de penser à combiner l'approche centrée sur la propagation de l'état de l'art (PGLNS) avec l'approche que nous proposé dans ce chapitre. Le schéma de combinaison retenu se base sur une combinaison simple des algorithmes de calcul de voisinage proposés

dans chaque approche (c.-à-d. chaque voisinage est appliqué séquentiellement jusqu'à ce qu'une nouvelle solution soit trouvée) : `exp-obj`, `exp-cft`, *propagation-guided neighborhood*, *reverse propagation-guided neighborhood*, *random propagation-guided neighborhood*. Les résultats produits sont sans appel. La combinaison des approches trouve des solutions équivalentes, ou meilleures, dans 69.4 % des instances traités (34 sur 49). Sur les 16 autres instances, elle est toujours classée seconde, presque toujours derrière EBLNS. De plus, elle est plus performante que PGLNS dans 77.5% des instances traitées, et fait mieux que EBLNS dans 65.3 % des instances traitées. En général, combiner les voisinages de PGLNS et EBLNS est profitable en terme de qualité des solutions, mais également en terme de stabilité. Dans 58.8 % des instances mieux résolues avec l'approche combinée, l'amplitude est inférieure à 5 %, ce qui signifie que quasiment toutes les résolutions aboutissent à la même dernière solution.

4.3. Conclusion et perspectives

Ce chapitre a résumé une approche flexible et facile à mettre en œuvre de la technique de recherche à voisinage large (LNS). Deux stratégies basée sur le calcul d'une explication liée à la meilleure solution courante ont été définies : expliquer d'une part l'échec provoqué par l'introduction d'une coupe liée à l'amélioration de l'objectif et, d'autre part, expliquer le retrait des valeurs de la variable objectif ayant conduit à la solution courante. Les résultats obtenus sont plutôt favorables par rapport à l'état de l'art mais une analyse plus fine de ces derniers nous ont permis de montrer une certaine forme de complémentarité entre notre approche et l'état de l'art. L'évaluation pratique d'une combinaison "simple" des deux approches nous a montré le bien fondé de notre intuition avec des résultats assez convaincants. Les évaluations plus fines qui n'ont pas été rapportées dans ce manuscrit ont montré cependant une forte dépendance de notre approche vis-à-vis de la stratégie de recherche (comment les décisions sont prises) et de la propagation (dans quel ordre le filtrage est effectué). En effet, l'explication liée à une solution n'est pas unique et est entièrement dépendante de ces deux derniers facteurs. Typiquement, utiliser des contraintes globales au sein du modèle, et pour peu qu'elles soient expliquée de manière pertinente, entraine un effet direct sur la qualité du voisinage calculé en sortie. De la même manière, changer la stratégie de recherche modifie là encore la forme de l'explication produite même si la solution proposée reste la même.

En termes de production logicielle, l'une des contributions fondamentales de ce travail réside dans l'implémentation d'un solveur de contraintes nativement expliqué, Choco-3.1.0 [TEA 13a, PRU 13]. À la différence de PaLM [JUS 00a], qui était une extension d'une précédente version de Choco, cette version du système Choco fournit un moteur d'explications entièrement natif et totalement débrayable.

Enfin, l'utilisation d'explications entraine un surcoût réel dans l'approche LNS malgré notre gestion "paresseuse" de ces dernières. Dans le cadre de nos expérimentations, nous avons mis en place une base de *NoGoods* alimentée au fur et à mesure de la progression de la recherche. Cette technique s'est avérée rarement compétitive avec les autres approches présentées dans ce chapitre, et ceci pour de simples raisons d'efficacité. En effet, la propagation devenait de plus en plus lourde à mesure que la base de NoGoods était alimentée. Cette observation sur l'ajout de techniques "intelligentes" mais coûteuses algorithmiquement va de pair avec le constat récurrent que l'efficacité de la recherche à voisinage large repose grandement sur le nombre de voisins qu'il est possible de tester dans une unité de temps. Plus ce nombre sera élevé, et plus grande sera sa chance de réussite. Dans une telle situation, alourdir le calcul des voisinages pour trouver une solution améliorante n'est pas forcément gage de performance. On retrouve ici la notion de compromis récurrent en Programmation par Contraintes : faut-il être plus exhaustif ou plus intelligent dans la stratégie de résolution ? C'est un débat que l'on retrouve à tous les étages des systèmes de contraintes (de la recherche à la propagation, en passant par le filtrage). La prochaine partie de ce manuscrit va s'attacher, dans le périmètre de cette question, à synthétiser les travaux que nous avons effectués autour des algorithmes de filtrages dans le cadre des contraintes globales.

DEUXIÈME PARTIE

Utilisation des contraintes globales

L'étude des algorithmes de propagation et de recherche constitue, avec la modélisation, la très grande majorité des travaux actuels dans la communauté. Cependant, les limites d'un paradigme purement déclaratif où tout problème pourrait être décomposé en contraintes "simples" reste encore aujourd'hui un vœu pieux cher à notre communauté. En pratique, il faut très souvent pousser l'analyse et l'optimisation des modèles au niveau des algorithmes de filtrages des contraintes elles-mêmes afin de capturer une part significative de la combinatoire des problèmes. Cette analyse pragmatique du comportement des systèmes de programmation par contraintes a donné naissance aux *contraintes globales* et à tout un nouveau champ d'investigations lié au monde de la recherche opérationnelle.

Les contraintes globales [BES 03] permettent d'exprimer la sémantique d'une relation logique au sein d'un sous-ensemble, de taille quelconque, des variables mises en jeu dans un CSP. La contrainte globale la plus connue est, sans conteste, la contrainte de différence [RÉG 94] `allDifferent`$(v_1, v_2, \ldots, v_n)$ qui exprime que toutes les valeurs affectées aux variables $v_1, v_2, \ldots, v_n$ doivent être deux à deux distinctes. Cette dernière peut aussi être exprimée par une représentation basée sur $\frac{n \times (n-1)}{2}$ contraintes de différence binaires du type $v_i \neq v_j$. Mais alors, qu'est-ce qui différencie une contrainte globale (comme `allDifferent`) de sa contrepartie basée sur une reformulation par des contraintes d'arité plus faible ? Il s'agit principalement de l'aspect *opérationnel*. En effet, le rendement effectif de l'algorithme de filtrage qui lui est associé dépasse largement la capacité déductive de "simples" contraintes de différences binaires. D'une manière plus générale, c'est le comportement recherché pour ce type de contraintes dont l'efficacité, en termes de temps de calcul, constitue la principale limite. Ainsi, comme c'est le cas pour la contrainte `allDifferent`, si un algorithme de filtrage polynomial (basé sur le calcul d'un couplage de cardinalité maximum dans un graphe bipartie) supprime du domaine de chaque variable toute valeur n'appartenant à aucune solution compatible alors on dira que cet algorithme effectue un *filtrage complet* des domaines des variables. Cependant, beaucoup de contraintes globales servent à modéliser des problèmes qui sont intrinsèquement NP-complets, ce qui, bien entendu, entraîne que seul un *filtrage partiel* des domaines des variables peut être effectué au travers d'un algorithme polynomial. C'est, par exemple, le cas de la contrainte `nvalue`, qui permet de restreindre le nombre de valeurs distinctes assignées à un ensemble de variables et dont l'un des aspects peut se réduire au problème de *minimum hitting set* [BEL 01, BES 04]. Finalement, le caractère "global" de

ces contraintes permet de simplifier le travail du solveur par le fait qu'il fournit toute ou partie de la structure du problème à résoudre. Lorsque l'on parle de contrainte globale, l'aspect *algorithmique*, s'intéressant à la complexité en temps et en espace, est aussi à prendre en compte. Supposons que l'algorithme de filtrage associé à une contrainte globale soit "complet" et qu'il existe une reformulation de cette contrainte strictement équivalente au niveau opérationnel, qu'en est-il au niveau algorithmique ? Cette question implique d'être à même de comparer le coût de l'algorithme de filtrage de la contrainte globale, au coût global du modèle décomposé au sein de l'algorithme de propagation.

Pourtant, il faut bien comprendre que les contraintes globales constituent une singularité dans le paradigme de la programmation par contraintes. En effet, elles sont d'une certaine manière antinomique avec les objectifs initiaux de la programmation par contraintes : décomposer un problème combinatoire en briques simples (les contraintes) et déléguer à un mécanisme générique (l'algorithme de propagation) la gestion du caractère intrinsèquement combinatoire du problème à résoudre. Pourtant, la nécessité de rendre le paradigme efficient, combinée aux limites connues des algorithmes de propagation, a rapidement mené les recherches sur la piste d'encapsuler une partie de la combinatoire des problèmes en une seule structure : les contraintes globales. Cependant, est-il acceptable d'encapsuler un sous-problème du problème à résoudre, potentiellement lui-même NP-complet, au sein d'un algorithme de filtrage entièrement dédié ? À cette question, la communauté a tantôt répondu par l'affirmative, dans une course à la performance et à la diffusion industrielle de la programmation par contraintes, et tantôt par la négative, avec un retour aux fondamentaux et à la décomposition en contraintes "simples". L'objectif de cette partie n'est pas de pencher pour un côté plus que l'autre, mais de montrer comment et sous quelles conditions les contraintes globales peuvent prendre une place appropriée au sein du paradigme. Pour ce faire nous prendrons trois points de vues distincts :

1) Chapitre 5 : la présentation des contraintes de partitionnement de graphes et leur nécessité dans le traitement des problèmes de satisfaction liés aux graphes.

2) Chapitre 6 : la présentation d'une contrainte globale "générique"permettant de modéliser une large famille de sous-problèmes liés au comptage.

3) Chapitre 7 : une analyse en moyenne du comportement de la contrainte globale `allDifferent` et les effets positifs qu'une telle étude pourrait avoir sur son utilisation

au sein d'un outil de programmation par contrainte.

Dans ce manuscrit, un certain nombre de travaux récents basés sur l'utilisation des graphes en programmation par contraintes et plus particulièrement dans le cadre des contraintes globales basées sur les graphes ne seront pas rapportés. On y trouvera une étude approfondie des heuristiques de recherche dans le cadre des contraintes de partitionnement en chemins (chemin Hamiltonien, TSP, etc.) dans les graphes [FAG 14b]. On retrouvera aussi une étude sur la décomposition systématique de contraintes globales raisonnant sur la microstructure du réseau (sous forme de graphe) associé à la contrainte. L'objectif est alors de diminuer le coût global de l'algorithme de filtrage associé à la contrainte (par un raisonnement local) afin de se concentrer sur les morceaux du réseau de la contrainte qui sont réellement impactés par les derniers changements de l'état des domaines [FAG 14a].

Chapitre 5

De la nécessité des contraintes globales : le cas des graphes

Sommaire

Dans la récente histoire de la programmation par contraintes, les contraintes globales ont constitué une singularité qui a assuré une certaine pérennité à ce paradigme. Comme nous l'avons dit en introduction les contraintes globales constituent un outil puissant tant en termes de modélisation, qu'en termes de résolution. Aujourd'hui encore, les contraintes globales les plus utilisées sont basées sur une utilisation intensive de concepts issus de la théorie des graphes. Nous pourrons retenir principalement les contraintes de différence (`allDifferent` [RÉG 94]), les contraintes de cardinalité (`globalCardinality` [RÉG 96]) et les contraintes à base d'automates (`regular` [PES 04], `costRegular` [DEM 06] et `multicostRegular` [MEN 09]). Plus généralement, le lecteur pourra se référer au catalogue de contraintes [BEL 10a] pour se faire une idée plus complète des propriétés de graphes utilisées dans les contraintes globales. Réciproquement, certains problèmes difficiles modélisés et traités à l'aide de la théorie des graphes ont été abordés avec succès en programmation par contraintes et plus particulièrement avec l'apparition des contraintes globales. Il s'agit principalement de contraintes autour de l'appariement de graphes (isomorphisme de graphe [SOR 04] et de sous-graphe [ZAM 07]), de la recherche de chemins dans des graphes [QUE 06, SEL 03], ou encore d'arbres couvrants de coût minimum [RÉG 08].

Les contraintes de partitionnement de graphe par des arbres ont été intensivement étudiées depuis dix ans, le lecteur pourra consulter si nécessaire une synthèse faite dans l'ouvrage [LOR 11] autour de la théorie et de quelques applications pour cette famille de contraintes. La thèse en cours de Jean-Guillaume Fages a pour objectif de venir enrichir les travaux de cet ouvrage en s'attaquant aux problèmes d'optimisation liés aux partitionnement de graphes. Plus particulièrement, cette thèse s'intéresse aux interactions entre les méthodes hybrides pour l'optimisation, issues de la recherche opérationnelle (par exemple, relaxation lagrangienne, relaxation linéaire, etc.), les algorithmes de filtrage proposés par les contraintes globales, et les heuristiques que l'on peut mettre en œuvre en programmation par contraintes. Ce chapitre ajoute une pierre à cet édifice en répondant à une des questions restée en suspens avant le stage de master de Jean-Guillaume Fages et qui m'a animé durant mon propre doctorat : Peut-on assurer l'arc-consistance de la contrainte `tree` en temps linéaire sur le nombre d'arcs du graphe en entrée ? C'est la réponse à cette question que nous avons choisi de rapporter dans ce chapitre [FAG 11].

La contrainte `tree` assure le partitionnement d'un graphe $G = (V, E)$ en un ensemble de L arbres sommets disjoints, où $|V| = n$, $|E| = m$ et L est une variable entière. Dans [BEL 05], nous avons montré que l'arc-consistance généralisée (GAC) pouvait être atteinte avec une complexité en temps au pire cas $O(nm)$ alors que le test de faisabilité pouvait lui être effectué en temps linéaire $O(n+m)$. Ainsi, l'algorithme de filtrage était fortement pénalisé par le calcul des points forts d'articulation qu'on ne savait pas calculer en temps linéaire au moment de ces travaux. Reposant sur les travaux de [ALS 99, BUC 98], Italiano et. al [ITA 10] ont proposé un algorithme linéaire pour les calculer en montrant le lien direct entre la notion de point fort d'articulation et celle de sommet dominant dans les graphes de flots. Cette nouvelle contribution nous a donc permis de dériver assez naturellement un algorithme de filtrage linéaire pour la contrainte `tree`.

Ce chapitre s'articulera en cinq sections, la première rappelant les définitions classiques de la théorie des graphes nécessaires à la compréhension de la règle de filtrage principale pour la contrainte `tree`. Ensuite, la section 5.2 présentera très rapidement l'état de l'art autour des contraintes de partitionnement de graphe du point de vue de la programmation par contraintes. La section 5.3 présentera ensuite les conditions nécessaires et suffisantes pour aboutir à un algorithme de filtrage complet et linéaire sur le nombre d'arêtes du graphe. Pour terminer, une rapide évaluation démontrera l'apport d'une telle contrainte par rapport aux résultats de l'état de l'art avant de conclure. Pour rappel tous les détails techniques et en particulier les preuves relatives aux différentes propositions sont disponibles dans [FAG 11].

5.1. Préliminaires

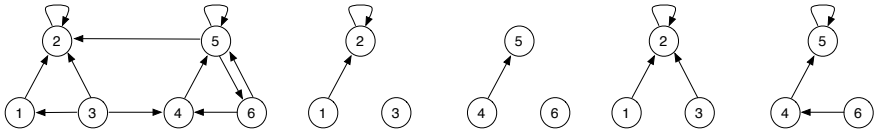
Un *graphe* $G = (V, E)$ est l'association d'un ensemble de sommet V et d'un ensemble d'arêtes $E \subseteq V^2$. Une arête $(x, y) \in E$, pour $x, y \in V$, signifie que les sommets x et y sont connectés dans G. Un graphe *orienté* est un graphe où les arêtes sont orientées d'un sommet source vers un sommet destination, on le notera $G = (V, A)$. Alors, un arc $(x, y) \in A$, pour $x, y \in V$, est une connexion orienté de x vers y, mais rien ne permet d'affirmer que la connexion de y à x existe.

Un *arbre* $T = (V, E)$ graphe non-orienté, acyclique. La version orienté du concept d'arbre est appelée *anti-arborescence* et correspond à un graphe orienté $T = (V, A)$

tel que tout sommet $v \in V$ possède exactement un successeur $w \in V$ et une unique racine $r \in V$, telle que $(r, r) \in A$, et il existe un chemin élémentaire de tout v à r. Dans la suite de chapitre nous nous placerons dans le contexte des anti-arborescences même si nous utiliserons le terme d'arbre par abus de langage.

Dans la suite de ce chapitre, nous allons utiliser de manière intensive toutes les notions liées à la connexité et à la forte connexité dans les graphes : nous parlerons donc librement de point d'articulation, point fort d'articulation, graphe de flot, sommet dominant en supposant que le lecteur connait ces définitions. Dans le cas contraire, ce dernier pourra se référer à l'article sur lequel repose ce chapitre [FAG 11] ou l'état de l'art sur la théorie des graphes [BER 70, GON 85]. Nous pouvons donc maintenant rappeler la définition centrale du partitionnement de graphe par des arbres qui fait l'objet de notre attention dans ce chapitre :

Définition 5.1. *Given a digraph $G = (V, A)$, a* tree partition *of G is a subgraph $P = (V, A_2)$, $A_2 \subseteq A$, such that each connected component is a tree.*



(a) Enveloppe $G_E = (V, A_E)$ (b) Noyau $G_K = (V, A_K)$ (c) Solution ($A_E = A_K$)

Figure 5.1. *A graph variable associated with a directed graph $G = (V, A)$*

Ainsi, nous pouvons caractériser proprement le partionnement d'un graphe en arbres : Soit $P = (V, A_2)$ un sous-graphe orienté du graphe orienté $G = (V, A)$, et $R = \{r | r \in V, (r, r) \in A_2\}$ un ensemble de sommet de A_2, on dira que P forme une partition de G en arbres si et seulement si chaque sommet de V a exactement un successeur dans P et, pour chaque sommet $v \in V$, il existe un sommet $r \in R$ tel qu'un chemin de v à r existe dans P. En programmation par contraintes une *solution* acceptante pour la contrainte `tree` est une partition en arbres d'un graphe orienté $G = (V, A)$ donné en entrée. Pour ce faire, G est modélisé à l'aide d'une *variable graphe* GV à laquelle nous associons un domaine défini par deux graphes : l'*enveloppe* $G_E = (V, A_E)$, contenant tous les arcs qui appartiennent potentiellement à au moins une solution (Figure 5.1(a)), et le *noyau* $G_K = (V, A_K)$, contenant tous les

arcs qui apparaissent dans toutes les solutions (Figure 5.1(b)). Nous retiendrons que la relation $A_K \subseteq A_E \subseteq A$ est un invariant associé à la variable graphe GV. *Filtrer* ce type de variables revient à supprimer des arcs de A_E et ajouter des arcs dans A_k. GV sera dite *instanciée* lorsque $A_E = A_K$ (Figure 5.1(c)). Un critère d'échec sur la variable sera l'inégalité $|A_E| < |V|$.

Définition 5.2. *Étant donné une variable graphe instanciée, on dira qu'elle est une partition en arbre de son graphe associé G si et seulement si son noyau G_K est une partition en arbre de G. Une variable graphe partiellement instanciée contient encore au moins une solution acceptante pour la contrainte* `tree` *si et seulement si il existe une partition en arbres de G_E.*

5.2. Autour de la contrainte `tree`

Cette section pose le cadre des travaux existant autour de la contrainte `tree` et des problèmes de partitionnement dans les graphes en général. Nous commencerons par proposer une décomposition naïve [LOR 11], qui n'assure aucun niveau de consistance. Ensuite, nous irons synthétiser différentes contribution de l'état de l'art qui peuvent servir dans la modélisation de ce problème. Enfin, nous conclurons ce cadre général par le rappel de l'algorithme de filtrage assurant la GAC [BEL 05].

5.2.1. *Une décomposition naïve*

Une approche par décomposition du problème de partitionnement de graphe par des arbres peut être proposée. Il faut pour cela définir une variable entière L caractérisant le nombre d'arbres autorisés dans la partition. Ensuite, soit $G = (V, E)$ un graphe d'ordre n, nous associons à chaque sommet i une variable entière v_i définissant les successeurs de i dans G, une variable entière r_i définissant le rang de i dans une solution et une variable booléenne b_i caractérisant la propriété de racine pour le sommet i. On notera que la variable r_i est introduite car les sommets composants un arbre peuvent être ordonnés par un tri topologique de manière unique, la valeur affectée à r_i donne la hauteur de i dans l'arbre auquel il appartient (afin de prévenir la création de circuits). Ainsi, le modèle du problème de partitonnnement de graphe par des arbres

peut être exprimé par :

$$v_i = j \wedge i \neq j \Rightarrow r_i < r_j,\ \forall i \in [1; n] \tag{5.1}$$

$$b_i \Leftrightarrow v_i = i,\ \forall i \in [1; n] \tag{5.2}$$

$$L = \sum_{i=1}^{n} b_i \tag{5.3}$$

La correction du modèle se prouve en considérant les trois cas suivants : (1) le modèle n'admet pas de solution contenant plus de L arbres bien formés, (2) le modèle n'admet pas de solution contenant moins de L arbres bien formés, (3) le modèle n'admet pas une solution contenant un circuit et (4) le modèle n'admet pas une solution contenant un sommet isolé sans boucle. Soit $G = (V, E)$ un graphe orienté d'ordre n et F une partition de G en α arbres bien formés :

– pour le cas (1), supposons que $\alpha > L$ alors si $\alpha > \sum_{i=1}^{n} b_i$ cela signifie qu'il existe plus d'arbres bien formés dans F que de sommets qui soient des racines potentielles, ce qui est impossible d'après la contrainte 5.2;

– pour le cas (2), supposons que $\alpha < L$ alors si $\alpha < \sum_{i=1}^{n} b_i$ cela signifie que F contient plus de boucles que d'arbres alors certains contiennent plus d'une boucle, mais comme chaque sommet a exactement un successeur : c'est une contradiction;

– pour le cas (3), s'il existe $f \in F$ tel que f contient un circuit alors, il existe deux sommets i et j dans f tels que l'on a un chemin élémentaire de i à j et un chemin élémentaire de j à i et par conséquent d'après la contrainte (5.1) on a $r_i < r_j$ et $r_j < r_i$: c'est une contradiction;

– la preuve du cas (4) est évidente car chaque variable doit être fixée à une valeur, ce qui équivaut à dire que chaque sommet doit avoir exactement un successeur.

5.2.2. *État de l'art*

Différents travaux connexes peuvent être utilisés afin de modéliser le problème de partitionnement de graphe par des arbres. On retiendra particulièrement les efforts produits pour le cas des chemins dans [SEL 03] et plus récemment dans [QUE 06]. Dans ce dernier travail, les variables de graphes sont utilisées afin de décrire une contrainte d'accessibilité entre sommets, nommée `DomReachability`, dans un modèle

de flot. Ainsi, cette contrainte assure que tous les sommets sont accessibles depuis une sommet source défini. La règle de filtrage principale repose principalement sur le concept fondamental de dominance dans le graphe de flot qui sera repris dans la suite. La limite de cette approche concerne le niveau de consistance atteint qui n'assure pas la GAC malgré une complexité temporelle au pire cas $O(nm)$.

Les travaux dédiés au partitionnement de graphe par des arbres prennent leur origine dans [BEL 05] et on fait l'objet de nombreuses extensions synthétisées dans l'ouvrage [LOR 11]. La contrainte initiale `tree` repose sur deux algorithmes : Le premier permet de vérifier l'existence d'une solution en temps linéaire $O(n+m)$ et le second permet de filtrer tout arc n'appartenant à aucune solution en $O(nm)$ (la GAC est donc assurée). Cet algorithme de filtrage se décompose simplement en deux parties : l'une relative au filtrage des bornes du nombre d'arbres, l'autre relative au filtrage des arcs dans le graphe. Le filtrage relatif au nombre d'arbres compatible pour la partition est assez évident. Il consiste d'une part à s'assurer que le domaine de la variable du nombre d'arbre compatibles est dimensionnée avec le nombre minimum (respectivement maximum) d'arbres qu'il possible de créer pour le graphe considéré. Le filtrage de cette variable se résume à considérer d'une part le nombre de composantes fortement connexes *puits* dans le graphe pour évaluer le nombre minimum d'arbres compatible, et d'autre part, le nombre de *racines potentielles*. Le filtrage structurel (maintenant la propriété que le graphe solution est une forêt) se base sur trois règles simples. La première s'assure que chaque composante fortement connexe puits dispose d'une racine potentielle valide. La seconde s'assure que chaque composante fortement connexe non puits peut atteindre une composante fortement connexe puits. Enfin, la troisième s'assure qu'à l'intérieur de chaque composante fortement connexe, pour chaque point fort d'articulation p existant, l'affectation d'un successeur à p ne créé pas une composante fortement connexe puits sans racine potentielle valide. Le goulot d'étranglement au niveau de la complexité restait jusqu'au travaux de [BEL 05] le calcul des points fort d'articulation afin d'assurer la complétude du filtrage. ceci conduisait donc à une complexité temporelle pour l'algorithme de filtrage GAC au pire cas en $O(nm)$.

La suite de l'histoire a voulu que les travaux menés par Italiono et. al. [ITA 10] permettent d'établir l'équivalence entre point fort d'articulation et dominance dans les graphes de flots. Ceci conduisant à un algorithme de calcul linéaire en $O(m+n)$ au

pire cas. Cependant, la dérivation d'un algorithme de filtrage lui-même linéaire n'est pas aussi directe que l'on pourrait le penser, en particulier pour effectivement maintenir le modèle de graphe de flot associé au graphe courant afin de filtrer efficacement les arcs incompatibles. c'est la contribution que nous allons synthétiser dans la suite.

5.3. Filtrage linéaire pour la contrainte `tree`

Cette section s'attache à décrire l'amélioration de l'algorithme de filtrage GAC pour l'a contrainte `tree`. Nous considèrerons dans la suite une variable graphe partiellement instanciée $GV = (G_E, G_K)$ qui représente un sous-graphe du graphe orienté fournit en entrée $G = (V, A)$. Nous noterons $G_E = (V, A_E)$, $G_K = (V, A_K)$ et $A_K \subseteq A_E \subseteq A$.

5.3.1. *Conditions de faisabilité et de filtrage*

Pour une variable graphe GV, associée à un graphe orienté G, partiellement instanciée, on dira qu'il existe une partition de G en arbres si et seulement si pour chaque sommet $v \in V$, les deux conditions de faisabilité suivantes sont vérifiées :

– $|\{(v, w) \mid (v, w) \in A_E\}| \geq 1$ et $|\{(v, w) \mid (v, w) \in A_K\}| \leq 1$

– Il existe un chemin de v à une racine potentielle $r \in V$ dans G_E.

De cette condition de faisabilité pour une contrainte `tree`, on peut dériver les deux propositions suivantes, l'une caractérisant l'ensemble des arcs incompatibles avec une partition en arbres de G, l'autre caractérisant les arcs appartenants à toute partition en arbres de G.

Proposition 5.1. *Pour une variable graphe GV, associée à un graphe orienté G, il existe une partition en arbres de G alors, un arc $(x, y) \in A_E$, $x \neq y$, est incompatible avec toute solution si et seulement si une des conditions suivantes est vérifiée :*

1) Il existe un sommet $w \in V, w \neq y$, tel que $(x, w) \in A_K$,

2) Tout chemin dans G_E depuis y à une quelconque racine potentielle $r \in V$ traverse le sommet x.

Proposition 5.2. *Pour une variable graphe GV, associée à un graphe orienté G, si chaque arc incompatible a été supprimé de GV alors, un arc $(x, y) \in A_E$, $x \neq y$,*

appartient à toute solution si et seulement si tout chemin dans G_E depuis x à une quelconque racine potentielle traverse l'arc (x, y).

5.3.2. *Algorithme de filtrage*

Nous nous plaçons dans le contexte "simplifié"où la variable graphe GV est représentée par deux tableaux de listes : l'un pour les successeurs et l'autre pour les prédécesseurs des sommets. La liste positionnée à l'index i du tableau des successeurs représente donc les successeurs du sommets $i \in V$. Dans la suite, nous simplifierons la notation de certaines quantités : $n = |V|$, $m = |A|$, $m_E = |A_E|$ avec $m_E \leq m$ (Section 5.1).

Existence d'une solution. Nous pouvons tout d'abord dériver un algorithme vérifiant l'existence d'au moins une solution pour la contrainte `tree` sur un graphe orienté G. Vérifier que les domaines de A_E et A_K sont compatibles est effectué en temps linéaire sur le nombre de sommets n en calculant simplement la taille de la liste de successeurs de chaque sommet, une fois pour G_E, une fois pour G_K. Soit R l'ensemble des racines potentielles, autrement défini $R = \{v | v \in V, (v, v) \in A_E\}$. Considérons, d'autre part, le graphe $G_{ES} = (V \cup \{s\}, A_E \cup S)$ où $s \notin V$, $A_E \cap S = \emptyset$ et $S = \bigcup_{r \in R}((r, s) \cup (s, r))$. Notons G_{ES}^{-1} le graphe inverse de G_{ES}, obtenu en reversant l'orientation des arcs de G_{ES}. Un parcours en profondeur d'abord (DFS) du graphe G_{ES}^{-1} depuis une racine potentielle s permet de vérifier si un sommet $v \in V$ est atteignable ou non depuis s par un chemin dans G_{ES}^{-1}. Alors, cette vérification permet effectivement de savoir si un sommet v peut atteindre une racine potentielle en utilisant un chemin dans G_E. Ainsi, on peut vérifier en temps linéaire sur le nombre d'arcs m_E qu'il existe un chemin de chaque sommet v à au moins une racine potentielle r. La complexité au pire cas est alors limitée à $O(n + m_E)$, avec $A_E \subseteq A$, $m_E \leq m$. On obtient donc un pire cas en fonction de la donnée d'entrée en $O(n+m)$.

Filtrage des arcs inconsistants. Nous proposons un algorithme de filtrage en $O(n+m)$ détectant tout arc de A_E n'appartenant à aucune solution pour la contrainte `tree`. Un tel algorithme repose sur les deux conditions énoncées par la Proposition 5.1 :

– *Condition 1:* Pour chaque sommet $v \in V$, soit v a un successeur dans G_K, soit v n'en a pas. Dans le premier cas, soit w un successeur dans G_K alors, la liste des successeurs de v dans G_E est nettoyée et l'arc (v, w) est ajouté dans G_E. Cet opération est effectuée en temps constant.

– *Condition 2:* Considérons le graphe inverse G_{ES}^{-1} précédemment introduit. Alors, la condition 2 de la Proposition 5.1 assure que le fait que l'arc $(x, y) \in G_E$ n'appartienne à aucune solution est équivalent au fait que x domine y dans le graphe de flot $G_{ES}^{-1}(s)$. Plusieurs algorithmes permettent de trouver les dominants dans un graphe de flot [ALS 99, BUC 98] avec une complexité en $O(n + m_E)$ au pire cas.

Alors nous avons la propriété qu'un sommet $p \in V$ domine un sommet $v \in V$ dans G_{ES}^{-1} si et seulement si p est un ancêtre de v dans l'arbre de dominance I calculé avec un des algorithmes de la condition 2. Une telle requête est effectuée en temps constant au prix d'un calcul initial en $O(n + m)$ (un parcours en profondeur mémorisant les numérotations pré- et post- ordre du graphe) et du stockage de l'arbre en $O(n)$ (une liste de taille n pour chaque numérotation). Au final, au plus m_E requêtes (une par arc de l'enveloppe) sont possibles conduisant à une complexité au pire cas de $O(n + m)$.

Dériver les arcs obligatoires. Pour rappel, forcer un (x, y) revient à l'ajouter dans le noyau A_K. La condition requise pour forcer un arc $(x, y) \in A_E$, donnée par la Proposition 5.2, est équivalente à dire que (x, y) appartient à toute solution si et seulement si l'arc (y, x) est une *arc dominant* dans le graphe de flot $G_{ES}^{-1}(s)$. L'état de l'art [ITA 10, QUE 06] propose de passer par l'insertion d'un sommet fictif dans chaque arc du graphe d'entrée et ensuite de calculer les sommets dominants. Ainsi, si un sommet fictif est désigné comme dominant, il sera naturel d'en déduire que l'arc associé dans le graphe d'origine est un arc-dominant. Ceci ne change rien à la complexité linéaire sur le nombre d'arcs du processus. Partant de l'assertion qu'un arc (y, x) est arc-dominant dans un graphe de flot $G(s)$ si et seulement si le sommet y est un dominant immédiat de x dans $G(s)$ et pour chaque prédécesseur p de x tel que $p \neq x$, x domine p dans $G(s)$, nous pouvons présenter une méthode alternative pour la détection des arcs obligatoires qui s'avère, en pratique, plus rapide et moins consommatrice de mémoire. Supposons que le l'algorithme filtrage a été appliqué précédemment. Alors, l'arbre de dominance I, du graphe G_{ES}^{-1}, est déjà connu et le pré-calcul de la relation sur les ancêtres dans I est lui aussi connu. Ainsi, un parcours en profondeur d'abord (DFS) du graphe G_{ES}^{-1} est effectué depuis le sommet s. Pour chaque arc (y, x), tel que $(x, y) \in I$ et $(x, y) \notin A_K$, pour chaque prédécesseur p de x, une requête pour savoir si x est un ancêtre de p dans I est appelée.Si l'une de ces requêtes répond par la négative alors l'arc (y, x) n'est pas arc dominant de $G_{ES}^{-1}(s)$.

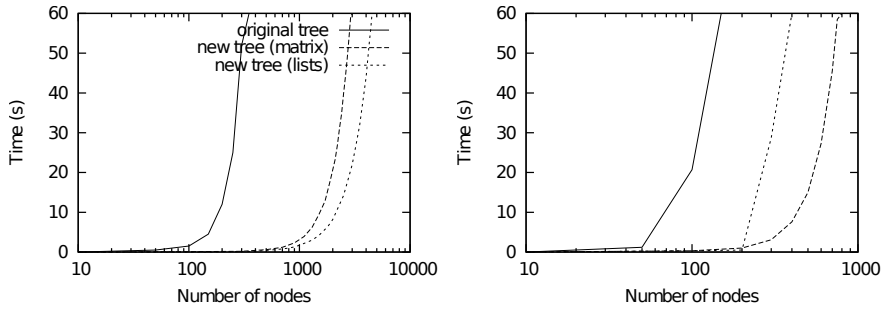
Sinon, l'arc (x, y) peut être ajouté dans A_K. Cet algorithme appelle $O(m)$ requêtes coûtant chacune du temps constant.

Le lecteur avertit pourra se questionner sur l'intérêt de caractériser les arcs obligatoires alors que l'objectif de l'algorithme de filtrage reste la détection des arcs impossible. Cependant, si dans notre cas chaque sommet possède exactement un successeur dans une solution, ce n'est pas forcément le cas dans un cadre plus général (cas non orienté, etc). Alors, l'identification des arcs obligatoires qui n'est pas dérivée directement du filtrage des arcs impossibles devient une information intéressante pour faire avancer le processus de résolution.

5.4. Évaluation

Les évaluations que nous avons effectuées ont pour objectif de valider l'apport de ce nouvel algorithme de filtrage. Les comparaisons sont effectuées par rapport à la décomposition de la contrainte proposée en Section 5.2.1 et l'algorithme de filtrage original Section 5.2.2. Chaque algorithme de filtrage est inclus dans un modèle qui recherche un partition en arbres d'un graphe connexe généré aléatoirement (à partir de l'ordre n et de la moyenne des degrés sortant pour chaque sommet $\overline{d^+}$). Nous testons aussi deux représentations possibles de la structure de graphe : la première basée sur une représentation matricielle, la seconde par une liste d'adjacente. La stratégie de recherche utilisée consiste à sélectionner aléatoirement un arc et à le rendre obligatoire (c.-à-d. à l'ajouter dans A_K). Tous les algorithmes ont été développés dans la librairie de contrainte Choco avec une machine Mac mini 4.1, avec un processeur 2.4 Ghz Intel core 2 duo et 3 Go alloués à la machine virtuelle java.

Globalement les résultats sur des graphes d'ordre 50, 150 et 300, avec un degré moyen (nombre de successeurs moyen $\overline{d^+}$) fixé à 5, 20 et 50 , ont montré que l'approche par décomposition est totalement inadaptée avec 40% d'instances résolues en moins d'une minute. L'approche originale souffre d'une extrême sensibilité à la densité du graphe. Toutes les instances sont résolues pour un nombre moyen de successeurs inférieur à 50 mais au delà de 150, plus aucune instance ne sera résolue dans la limite. Le nouvel algorithme proposé semble totalement insensibles à la densité du graphe la totalité des instance sont résolues. Les graphiques de la Figure 5.2 montre clairement que le nouvel algorithme de filtrage de la contrainte `tree` surclasse totalement l'état de l'art. Des problèmes impliquant des graphes complets d'ordre supérieur



(a) Sparse graphs ($\overline{d^+} = 5$) (b) Complete graphs ($\overline{d^+} = n$)

Figure 5.2. *Evolution du passage à l'échelle en fonction de l'algorithme et de la structure de données employée.*

à 750 sommets ou des graphes très peu dense avec plus de 4500 sommets peuvent être résolus. Le choix de la structure de données est aussi très important en fonction de la densité du graphe. Ainsi, proposer un algorithme de filtrage adaptant la représentation de manière dynamique est quelque chose de crucial pour obtenir des performances remarquables : La densité critique semble se situer autour de $d_c = \frac{35}{n}$. Ainsi, lorsque $d < d_c$ la représentation par liste d'adjacente semble dominer alors que la représentation matricielle s'avère bien plus performante pour des graphes plus denses que cette limite. Le dernier jeu d'expérimentations effectuées relève du passage à l'échelle. Pour cela, avec une limite de résolution fixée à 2 minutes nous observons la taille maximale des graphes qui peuvent être proposés. Dans le cas des graphes très peu denses, représenté par liste d'adjacente, le nouvel algorithme permet d'attaquer des graphes de près de 6000 sommets, là où l'ancien algorithme bloque aux alentours de 350 sommets. Dans le cas des graphes denses, représentés par une matrice, notre algorithme attaque des graphes de l'ordre de 900 sommets, là où l'algorithme original bloque autour de 160 sommets.

5.5. Conclusion

Ce chapitre a résumé des travaux autour de l'algorithme de filtrage pour la contrainte `tree`. Une mise en œuvre de cet algorithme, avec une complexité en $O(m\alpha(n, m))$, a permis de d'améliorer drastiquement les performance de l'algorithme de filtrage original. Nous avons aussi montré expérimentalement l'importance du choix de la

structure de données en fonction du type de graphe attaqué. Finalement, notre seule "déception"est liée au caractère non incrémental de cet algorithme. Il est dû au fait qu'aujourd'hui, la propriété de dominance dans les graphes ne fait l'objet d'aucun raisonnement local et la communauté d'algorithmique de graphe ne semble pas encore avoir d'intérêt dans cette voie.

Enfin, ce chapitre, au travers des résultats proposés dans la section précédente, met en avant le cas particulier d'une contrainte de graphe pour laquelle toute décomposition en contraintes simples est une vaine tentative tant sa sémantique est liée à une propriété globale de la structure qu'elle représente (ici une partition en arborescences d'un graphe orienté). Il s'agit donc d'un cas d'école démontrant que sous certaines conditions les contraintes globales sont le seul moyen de pouvoir aboutir à une solution raisonnable pour un problème. Le constat mis en avant ici, se généralise à la quasi-totalité des contraintes impliquant un problème de graphe sous-jacent. Le phénomène est encore plus marqué lorsque nous prenons le point de vue de problèmes d'optimisation, tels que celui du voyageur de commerce ou sa généralisation au recouvrement par des arbres de poids minimaux sous contraintes de degré. En effet, l'encapsulation de techniques de relaxations (linéaires ou lagrangiennes), issues de la recherche opérationnelle, devient inévitable [BEN 12, FAG 12, FAG 13] pour espérer concurrencer les approches classiques de l'état de l'art. Une dernière singularité, prenant le paradigme de la programmation par contraintes à contre-pied, concerne l'utilisation de variables spécifiques au modèle des graphes. Là encore l'apparition de ce modèle de variables est directement guidée par un souci d'efficacité et d'expressivité (projet RNRT *ROCOCO* et [DOO 05]). La mise en œuvre efficace (ce qui n'est pas toujours le cas de l'état de l'art) est une tâche complexe qui nécessite un développement entièrement dédié et faisant abstraction de la représentation classique des variables en CP, qu'elles soient entières ou ensemblistes. Le lecteur intéressé consultera l'API de l'outil Choco afin de découvrir les stratégies efficaces de mise en œuvre.

Chapitre 6

Mutualiser les algorithmes de filtrage

Sommaire

Dans le zoo des contraintes globales, nombre d'entre elles sont liées à une manière opérationnelle de maintenir un certain nombre d'occurrences d'une propriété, à travers une variable de "comptage", sur une séquence de variables données. Il s'agit donc d'une forme récurrente de signature pour toute une famille de contraintes globales. Il est donc légitime de se poser la question d'une possible montée en abstraction d'un certain nombre d'algorithmes de filtrages jusqu'ici considérés comme indépendants.

Dans ce contexte, nous nous sommes intéressés, avec Nicolas Beldiceanu et Thierry Petit, à formaliser une forme de *méta-contrainte* globale sur une séquence binaire de contraintes SEQ_BIN(N, X, C, B) [PET 11] où N est la variable entière comptant les occurrences de la propriété, X est la séquence de variables entières observées et enfin, C et B sont deux contraintes binaires telles que chaque paire consécutive de variables de X satisfait B, et C modélise la propriété comptée sur la séquence X. Cette notion de contrainte modélisant la "propriété comptée" se formalise en se basant sur la notion de C*-stretch*, qui est une généralisation de la notion de *stretch* introduite dans [PES 01] et où la contrainte d'égalité est généralisée par la contrainte C. Alors, on dira que la contrainte SEQ_BIN est satisfaite si et seulement si les deux conditions suivantes peuvent être vérifiées : (1) N est égal au nombre de C-stretches sur la séquence X, et (2) B est vérifiée sur toute paire de variables consécutives de X.

Parmi les contraintes qui peuvent s'exprimer à l'aide de la contrainte SEQ_BIN, certaines sont plus significatives pour les systèmes réels, par ex., CHANGE [Cos97] (emploi du temps), SMOOTH [BEL 10a] (emploi du temps et ordonnancement), ou encore INCREASING_NVALUE [BEL 10b] (gestion des symétries dans les problèmes d'allocation de ressources). La principale contribution de chapitre est un algorithme de filtrage polynomial pour la contrainte SEQ_BIN, qui maintient l'arc consistance généralisée (GAC) suivant certaines hypothèses sur la forme des contraintes B et C. Cet algorithme est une généralisation de l'algorithme de filtrage introduit pour la contrainte INCREASING_NVALUE. Ainsi, étant donné n la taille de la séquence de variables X, d la taille maximum des domaines, et Σ_{Di} la somme de la taille des domaines, suivant certaines propriétés sur les contraintes B et C, nous montrerons que la GAC est atteignable avec une complexité en temps et en espace de $O(\Sigma_{\mathrm{Di}})$. Ces propriétés sont bien sûr satisfaites lorsque SEQ_BIN modélise les contraintes INCREASING_NVALUE, mais aussi différentes variantes de CHANGE. Pour ces contraintes, nous égalons ou dépassons les meilleurs résultats de l'état de l'art.

6.1. Travaux connexes et définition de SEQ_BIN

Dans cette section, nous commençons par formaliser la contrainte SEQ_BIN en utilisant une généralisation de la définition de *stretch* introduite par Pesant dans [PES 01]. Nous proposons ensuite deux exemples de contraintes classiques qui peuvent reformulées à l'aide de SEQ_BIN et pour lesquelles nous serons à même d'assurer un filtrage complet avec la meilleur complexité connue.

6.1.1. *Définition nécessaire*

Étant donné une séquence de variables $X = [x_0, x_1, \ldots, x_{n-1}]$ et la séquence de domaines $\mathcal{D}$ qui lui est associée dans un CSP, une séquence de variables $X' = [x_i, x_{i+1}, \ldots, x_j]$, $0 \leq i \leq j \leq n-1$ (resp. $i > 0$ ou $i < n-1$) est une *sous-séquence* (resp. une *sous-séquence stricte*) de X. on la note $X' \subseteq X$ (resp. $X' \subset X$). On note encore $A[X]$ une affectation de valeurs aux variables de X. Étant donné $x \in X$, $A[x]$ est la valeur de x dans $A[X]$. $A[X]$ est dite *valide* si et seulement si pour tout $x_i \in X$, $A[x_i] \in D(x_i)$. une *instanciation* $I[X]$ est une affectation valide de X. Étant donné $x \in X$, $I[x]$ est la valeur de x in $I[X]$. Étant donné la séquence X et deux entiers i, j tels que $0 \leq i \leq j \leq n-1$, on notera $I[x_i, \ldots, x_j]$ la projection de $I[X]$ sur $[x_i, x_{i+1}, , \ldots, x_j]$. Une *contrainte* $C(X)$ définie un sous-ensemble $\mathcal{R}_C(\mathcal{D})$ du produit cartésien des domaines $\Pi_{x_i \in X} D(x_i)$. Si X est une paire de variables, alors $C(X)$ est dite *binaire*. On notera vCw une paire de valeurs (v, w) qui satisfait la contrainte binaire C. $\neg C$ est la contrainte *opposée* à C. En d'autres termes, $\neg C$ définie la relation $\mathcal{R}_{\neg C}(\mathcal{D}) = \Pi_{x_i \in X} D(x_i) \setminus \mathcal{R}_C(\mathcal{D})$.

Avant d'introduire la contrainte SEQ_BIN, il est nécessaire de définir une généralisation de la notion de *stretch* introduite dans [PES 01] afin de caractériser une séquence de variables consécutives sur laquelle une même contrainte binaire sera satisfaite.

Définition 6.1 (C*-stretch*)**.** *Soit $I[X]$ une instanciation de la séquence de variables $X = [x_0, x_1, \ldots, x_{n-1}]$ et C une contrainte binaire quelconque. Une séquence de contraintes C sur $I[X]$, notée $\mathcal{C}(I[X], C)$, est satisfaite si et seulement si :*

– *Soit $n = 1$,*

– *soit $n > 1$ et $\forall k \in [0, n-2]$, $I[x_k] C I[x_{k+1}]$ est satisfaite.*

Un C-stretch *sur* $I[X]$ *est une sous-séquence* $X' \subseteq X$ *telle que les deux conditions suivantes doivent être satisfaites :*

1) $\mathcal{C}(I[X'], C)$ *est satisfaite,*

2) $\forall X"$ *tel que* $X' \subset X" \subseteq X$ *alors* $\mathcal{C}(I[X"], C)$ *n'est pas satisfaite.*

Pour appréhender cette définition, il faut considérer la sous-séquence de longueur maximale pour laquelle la contrainte binaire C est satisfaite entre les variables consécutives de la sous-séquence concernée. Grâce à cette généralisation, nous pouvons maintenant intorduire la contrainte SEQ_BIN(N, X, C, B). Considérons une variable N, une séquence de n variables $X = [x_0, x_1, \ldots, x_{n-1}]$ et deux contraintes binaires C et B et, une instanciation $I[N, x_0, x_1, \ldots, x_{n-1}]$, la contrainte SEQ_BIN sera satisfaite si et seulement si les deux conditions suivantes sont vérifiées :

– Pour tout $i \in [0, n-2]$, $I[x_i]\, B\, I[x_{i+1}]$ est satisfaite ;

– $I[N]$ est égal au nombre de C*-stretches* dans $I[X]$.

Dans nos travaux [PET 11], nous avons montré que la contrainte SEQ_BIN pouvait être reformulée à l'aide de la contrainte SLIDE$_2$ introduite par Bessière *et al.* [BES 08a]. Cette reformulation assurant elle-aussi un filtrage complet a cependant un coût algorithmique beaucoup plus élevé : $O(nd^3)$ contre $O(\Sigma_{\mathrm{Di}})$ (qui est borné au pire cas par $O(nd)$) dans notre cas. Pour finir avec l'état de l'art, il faut souligner que certains algorithme de filtrage dédiés peuvent être comparés à SEQ_BIN. Typiquement, la contrainte CHANGE [HEL 04, page 57] pour laquelle un algorithme dédié atteint la GAC en $O(n^3m)$, ou encore la contrainte INCREASING_NVALUE [BEL 10b] qui a exactement la même complexité que SEQ_BIN.

6.1.2. *Deux exemples*

La contrainte CHANGE a été introduite dans le cadre des problèmes de planification d'emploi du temps [Cos97] avec pour objectif de pour pouvoir limiter le nombre de changements de postes sur une période donnée. De plus, la relation entre la notion de stretch et le filtrage de la contrainte CHANGE a été mis en avant pour la première fois dans [HEL 04, page 64]. La contrainte CHANGE est définie par une variable N, une séquence de variables $X = [x_0, x_1, \ldots, x_{n-1}]$, et une contrainte binaire $C \in \{=, \neq, <, >, \leq, \geq\}$. Elle est dite satisfaite si et seulement si N est égal

au nombre de fois que la contrainte C est satisfaite sur des variables consécutives de la séquence X. La contrainte CHANGE peut être reformulée par la conjonction SEQ_BIN$(N', X, \neg C,\texttt{true}) \wedge [N = N' - 1]$, où `true` fait référence à la contrainte universelle. Une telle reformulation n'entraine aucune perte de filtrage (c.-à-d., il reste GAC) car le réseau de contrainte induit reste Berge-acyclique. De la même, le lecteur pourra avancer que CHANGE est formulable à l'aide contraintes à base d'automates, qu'il s'agisse deREGULAR [PES 04] ou de COST-REGULAR [DEM 06]. Cependant, dans le premier cas la complexité pour atteindre la GAC est en $O(n^2d^2)$ [BEL 10a, pages 584–585, 1544–1545] (où d représente le domaine de taille maximale), alors que la seconde reformulation basée sur COST-REGULAR ne permettra pas d'atteindre la GAC.

Notre second exemple porte sur la contrainte INCREASING_NVALUE qui est une version spécialisée de NVALUE [PAC 99]. Elle a été introduite dans l'objectif de casser les symétrie cachée dans les problèmes d'allocation de ressources dans les grands centres de données [BEL 10b]. INCREASING_NVALUE est définie par une variable N et une séquence de variables $X = [x_0, x_1, \ldots, x_{n-1}]$. Étant donné une instanciation, on dira que la contrainte INCREASING_NVALUE(N, X) est satisfaite si et seulement si N est égale au nombre de valeur distinctes affectées aux variables de la séquence X et pour tout entier $i \in [0, n-2]$, $x_i \leq x_{i+1}$. Une telle contrainte se reformule naturellement avec notre contrainte par : SEQ_BIN$(N, X, =, \leq)$.

6.2. Consistance d'une contrainte SEQ_BIN

Nous commençons par présenter comment calculer, pour une valeur quelconque dans le domaine d'une variable $x_i \in X$, le nombre minimum et maximum de C-stretches dans la séquence suffixe de X débutant sur la variable x_i (resp. la séquence préfixe de X terminant sur x_i) satisfaisant une chaîne de contraintes binaires du type B. Ensuite, nous introduirons différentes propriétés utiles pour obtenir une condition de faisabilité pour la contrainte SEQ_BIN, et une condition nécessaire et suffisante pour la règle de filtrage conduisant à un algorithme de filtrage GAC (présenté en Section 6.3).

De la définition de la contrainte SEQ_BIN, nous devons assurer que la chaîne de contraintes B est satisfaite sur la séquence de variables $X = [x_0, x_1, \ldots, x_{n-1}]$. On

dira qu'une instanciation $I[X]$ est B-*cohérent* si et seulement si soit $n = 1$, soit pour tout $i \in [0, n-2]$, nous vérifions $I[x_i]\, B\, I[x_{i+1}]$. Une valeur $v \in D(x_i)$ est dite B-*cohérent* pour x_i si et seulement si elle fait partie d'une instanciation B-cohérente. Alors, étant donné un netier $i \in [0, n-2]$, si $v \in D(x_i)$ est B-cohérent pour x_i alors il existe $w \in D(x_{i+1})$ tel que $v\, B\, w$. En conséquence, pour un domaine $D(x_i)$ donné, les valeurs qui ne sont pas B-coherente peuvent être supprimées puisqu'elles ne font parties d'aucune solution satisfaisant SEQ_BIN. Notre objectif est alors de calculer pour chaque valeur B-cohérente v dans le domaine de toute variable x_i le nombre minimum et maximum de C-*stretches* sur X.

Pour la suite nous allons formellement introduire quelques notations relatives au dénombrement des C-stretches sur une séquence de variables :

– $\underline{s}(x_i, v)$ (resp. $\overline{s}(x_i, v)$) is the minimum (resp. maximum) number of C-stretches within the sequence of variables $[x_i, x_{i+1}, \ldots, x_{n-1}]$ assuming $x_i = v$.

– $\underline{p}(x_i, v)$ (resp. $\overline{p}(x_i, v)$) is the minimum (resp. maximum) number of C-stretches within the sequence $[x_0, x_1, \ldots, x_i]$ under the hypothesis that $x_i = v$.

– Given $X = [x_0, x_1, \ldots, x_{n-1}]$, $\underline{s}(X)$ (resp. $\overline{s}(X)$) denotes the minimum (resp. maximum) value of $\underline{s}(x_0, v)$ (resp. $\overline{s}(x_0, v)$).

Alors, nous présentons ici la formule de calcul pour le nombre minimum (resp. maximum) de C-stretches.

Lemme 6.1. *Étant donné* SEQ_BIN(N, X, C, B) *avec* $X = [x_0, x_1, \ldots, x_{n-1}]$*, supposons que les domaines de* X *contiennent seulement des valeurs* B*-coherentes. Alors, étant donné* $i \in [0, n-1]$ *et* $v \in D(x_i)$*, on a :*

– *Si* $i = n-1$*:* $\underline{s}(x_{n-1}, v) = 1$ *et* $\overline{s}(x_{n-1}, v) = 1$.

– *Sinon :*

$$\underline{s}(x_i, v) = \min_{w \in D(x_{i+1})} \begin{pmatrix} \min_{[vBw] \wedge [vCw]}(\underline{s}(x_{i+1}, w)), \\ \min_{[vBw] \wedge [v \neg Cw]}(\underline{s}(x_{i+1}, w)) + 1 \end{pmatrix} \textit{and}$$

$$\overline{s}(x_i, v) = \max_{w \in D(x_{i+1})} \begin{pmatrix} \max_{[vBw] \wedge [vCw]}(\overline{s}(x_{i+1}, w)), \\ \max_{[vBw] \wedge [v \neg Cw]}(\overline{s}(x_{i+1}, w)) + 1 \end{pmatrix}$$

Étant donné une séquence de variables $[x_0, x_1, \ldots, x_{n-1}]$ telles que leurs domaines contiennent uniquement des valeurs B-cohérentes, alors, pour toute variable

x_i dans la séquence et toute valeur $v \in D(x_i)$, calculer $\underline{p}(x_i, v)$ (resp. $\overline{p}(x_i, v)$) est symétrique à $\underline{s}(x_i, v)$ (resp. $\overline{s}(x_i, v)$). Nous substituons $\min$ par $\max$ (resp. $\max$ by $\min$), x_{i+1} par x_{i-1}, et vRw par wRv pour toute contrainte $R \in \{B, C, \neg C\}$.

6.2.1. *Propriétés autour du nombre de C-stretches*

Le propriétés liant les valeurs du domaine de $D(x_i)$ avec le nombre minimum et le nombre maximum de C-stretches dans X sont résumés dans la suite. Nous considérons uniquement les valeurs B-cohérentes qui peuvent faire partie d'un instanciation valide de SEQ_BIN. Les trois points suivant synthétisent les trois propriétés majeures liées au Lemme 6.1. Considérons la contraintes SEQ_BIN(N, X, C, B), une variable $x_i \in X$ ($0 \leq i \leq n-1$), et deux valeurs B-coherentes $v_1, v_2 \in D(x_i)$:

1) Pour toute valeur B-cohérente v dans $D(x_i)$, pour une variable x_i, nous vérifions $\underline{s}(x_i, v) \leq \overline{s}(x_i, v)$;

2) Si $i = n-1$ ou s'il existe une valeur B-coherente $w \in D(x_{i+1})$ telle que $v_1 B w$ et $v_2 B w$, alors, $\overline{s}(x_i, v_1) + 1 \geq \underline{s}(x_i, v_2)$;

3) Considérons le cas où $i = n-1$, et le cas où il existe une valeur B-cohérente $w \in D(x_{i+1})$ telle que $v_1 \, B \, w$ et $v_2 \, B \, w$. Si un des deux cas est vérifié alors, pour tout $k \in [\min(\underline{s}(x_i, v_1), \underline{s}(x_i, v_2)), \max(\overline{s}(x_i, v_1), \overline{s}(x_i, v_2))]$, soit on a $k \in [\underline{s}(x_i, v_1), \overline{s}(x_i, v_1)]$, soit on a $k \in [\underline{s}(x_i, v_2), \overline{s}(x_i, v_2)]$.

6.2.2. *Propriétés sur les contraintes binaires*

La dernière propriété parmi les trois précédentes est centrale pour assurer un algorithme de filtrage GAC dérivé du comptage, pour chaque valeur B-cohérente d'un domaine, du nombre de *C-stretches*minimum et maximum dans une instanciation complète. Étant donné une contrainte SEQ_BIN(N, X, C, B), nous nous concentrons sur les contraintes du type B qui soient binaires monotones [Van 92]. Ceci nous permet alors de garantir la vérification de cette dernière propriété. Par exemple, les contraintes binaires telles que $<$, $>$, $\leq$ et $\geq$ vérifient la propriété de monotonie, ainsi que la contrainte universelle `true`.

Propriété 6.1. *Considérons une contrainte* SEQ_BIN(N, X, C, B) *telle que toute valeur non B-cohérente a été supprimée des domaines des variables de X. B est dite monotone si et seulement si pour toute variable $x_i \in X$, $0 \leq i < n-1$, pour toute valeur $v_1, v_2 \in D(x_i)$, il existe $w \in D(x_{i+1})$ telle que $v_1 B w$ et $v_2 B w$.*

6.2.3. *Filtrage de la contrainte* SEQ_BIN

Ainsi, partant de la Propriété 6.1, cette section dérive une relation d'équivalence entre l'existence d'une solution pour la contrainte SEQ_BIN et les domaines fourrant des variables de X et N. Sans perte de généralités, cette section considère que toute les valeurs non B-cohérentes ont été préalablement supprimées des domaines des variables de X. La définition initiale de la contrainte SEQ_BIN assure la condition nécessaire de faisabilité suivante :

Proposition 6.1. *Étant donné une contrainte* SEQ_BIN(N, X, C, B)*, si $\underline{s}(X) > \max(D(N))$ ou $\overline{s}(X) < \min(D(N))$ alors la contrainte* SEQ_BIN *lève une contradiction.*

Naturellement , le domaine $D(N)$ peut être filtré par $[\underline{s}(X), \overline{s}(X)]$, sachant qu'il n'est pas forcément un intervalle (des "trous" sont possibles) ou qu'il peut être strictement inclus dans $[\underline{s}(X), \overline{s}(X)]$. D'après cette proposition, toute valeur de N dans $D(N) \cap [\underline{s}(X), \overline{s}(X)]$ est GAC. Quoiqu'il en soit nous arrivons à la proposition centrale suivante :

Proposition 6.2. *Considérons une contrainte* SEQ_BIN(N, X, C, B) *telle que B soit monotone, avec $X = [x_0, x_1, \ldots, x_{n-1}]$. Pour tout entier k dans $[\underline{s}(X), \overline{s}(X)]$ il existe v dans $D(x_0)$ tel que $k \in [\underline{s}(x_0, v), \overline{s}(x_0, v)]$.*

Nous introduisons maintenant une propriété pour la contrainte SEQ_BIN(N, X, C, B) sur la continuité de comptage. Ainsi, on dira que la contrainte supporte un *comptage continu* si et seulement si pour chaque instanciation $I[X]$ avec k C-stretches, pour toute variable $x_i \in X$, modifier la valeur de x_i dans $I[X]$ ajoute à la valeur du nombre de C-stretches k, 0, $+1$ ou -1. On peut alors proposer une condition nécessaire et suffisante assurant l'existence d'une solution pour la contrainte.

Proposition 6.3. *Considérons une instance de la contrainte* SEQ_BIN(N, X, C, B) *supportant la propriété de comptage continu et telle que la contrainte B soit monotone, alors, on dira que* SEQ_BIN(N, X, C, B) *admet au moins une solution si et seulement si $[\underline{s}(X), \overline{s}(X)] \cap D(N) \neq \emptyset$.*

Pour un contrainte SEQ_BIN(N, X, C, B), la Proposition 6.3 permet de filtrer la variable N à partir de la séquence de variables X. Les Propositions 6.1 et 6.2 assurent que toute valeur restante dans $[\underline{s}(X), \overline{s}(X)] \cap D(N)$ peut-être étendue à au moins une solution satisfaisant SEQ_BIN. Nous pouvons maintenant présenter la proposition principale permettant de filtrer directement les variables X.

Proposition 6.4. *Pour une instance de la contrainte* SEQ_BIN(N, X, C, B) *supportant la propriété de comptage continu et telle que la contrainte B soit monotone, considérons une valeur v dans $D(x_i)$, $i \in [0, n-1]$. Les deux assertions suivantes sont équivalentes :*

1) la valeur v est B-cohérente et v est GAC pour la contrainte SEQ_BIN

2) $\begin{bmatrix} \underline{p}(x_i, v) + \underline{s}(x_i, v) - 1, \\ \overline{p}(x_i, v) + \overline{s}(x_i, v) - 1 \end{bmatrix} \cap D(N) \neq \emptyset$

La valeur "-1"dans les expressions $\underline{p}(x_i, v) + \underline{s}(x_i, v) - 1$ et $\overline{p}(x_i, v) + \overline{s}(x_i, v) - 1$ prévient de compter deux fois le même C-stretch sur une extrémité x_i des deux séquences $[x_0, x_1, \ldots, x_i]$ et $[x_i, x_{i+1}, \ldots, x_{n-1}]$.

6.3. Mise en œuvre pratique

Nous nous attachons tout d'abord à présenter, dans la section 6.3.1, les éléments permettant d'obtenir le principe d'un algorithme de filtrage générique pour la contrainte SEQ_BIN. Ensuite, la section 6.3.2 résumera les résultats obtenus dans [BEL 10b] pour le cas de l'utilisation de la contrainte INCREASING_NVALUE.

6.3.1. *Schéma d'un algorithme de filtrage générique*

Le schéma de l'algorithme de filtrage générique est dérivé de la condition nécessaire est suffisante introduite par la Proposition 6.4. Cette section détaille un schéma d'algorithme atteignant la GAC pour une instance de la contrainte SEQ_BIN(N, X, C, B) supportant la propriété de comptage continu et telle que la contrainte B soit monotone.

Si la contrainte $B \notin \{\leq, \geq, <, >, \texttt{true}\}$ alors l'ordre total $\prec$ assurant la monotonie de B n'est pas celui suivi par l'ordre naturel des entiers. Dans ce cas, si $\prec$ n'est pas connu, il est nécessaire de calculer un tel ordre en accord avec les valeurs

de $\cup_{i\in[0,n-1]}(D(x_i))$, et ce, avant la propagation initiale de la contrainte SEQ_BIN. Si l'on se place dans le contexte où les deux variables de la contrainte B prennent leurs valeurs dans $\cup_{i\in[0,n-1]}(D(x_i))$, alors, l'inclusion de l'ensemble des supports des valeurs nous assure que l'ordre reste préservé quand les domaines des variables de B ne contiennent pas toutes les valeurs de $\cup_{i\in[0,n-1]}(D(x_i))$. Ce réarrangement des valeurs dans les domaines peut être calculé à partir des supports de chaque valeur en $O(|\cup_{i\in[0,n-1]}(D(x_i))|log(|\cup_{i\in[0,n-1]}(D(x_i))|))$ [PET 11]. Alors, étant donné une séquence de variables X, l'algorithme est décomposé en quatre étapes :

1) Supprimer les valeurs non B-cohérentes des domaines des variables de X.

2) Pour toutes les valeurs des domaines des variables de X, calculer le nombre minimum et maximum de C-*stretches* sur les préfixes et les suffixes.

3) Ajuster les bornes de la variable N en accord avec le nombre minimum et maximum de C-*stretches* de X.

4) À partir des quantités de la phase 2 et de la Proposition 6.4, filtrer les valeurs B-cohérentes restantes.

Maintenant, dans le cas où $B \in \{\leq, \geq, <, >, \texttt{true}\}$, alors, la relation d'ordre $\prec$ est parfaitement connue. La complexité au pire cas de notre algorithme est alors déterminée par la phase 2 précédemment introduite. Ceci est lié au fait que lorsqu'un domaine $D(x_i)$ est analysé, le nombre minimum et maximum de C-stretches pour chaque valeur est recalculé entièrement, là où une gestion incrémentale permettrait d'éviter l'analyse de la totalité du domaine $D(x_{i+1})$ pour chaque valeur dans $D(x_i)$. La Propriété 6.2 synthétise le résultat de complexité. Une fois encore, nous restons concentré dans cette synthèse sur le cas du nombre minimum de C-stretches pour les suffixes, les autres cas étant symétriques.

Propriété 6.2. *Pour une instance de la contrainte* SEQ_BIN(N, X, C, B) *supportant la propriété de comptage continu telle que* $B \in \{\leq, \geq, <, >, \texttt{true}\}$ *et* $x_i \in X$, $0 \leq i < n-1$, *Si, pour tout* $v_j \in D(x_i)$, *le calcul de* $\min_{w\in D(x_{i+1})}(s(v_j, w))$ *peut être effectué en* $O(|D(x_{i+1})|)$ *alors, un filrage vérifiant la GAC peut être atteint pour la contrainte* SEQ_BIN *en* $O(\Sigma_{\mathrm{Di}})$ *à la fois pour la complexité temporelle et spatiale.*

6.3.2. *Résultats expérimentaux pour* INCREASING_NVALUE

Cette section présente une synthèse des résultats d'expérimentations menées autour de la contrainte INCREASING_NVALUE. Toutes ces expériences ont été réalisées à l'aide de la librairie de programmation par contraintes Choco, sur un processeur Intel Core 2 Duo 2.4GHz avec 4GB de RAM, et 128Mo alloués à la machine virtuelle Java.

Thus, when SEQ_BIN represents INCREASING_NVALUE, computing $\min_{w \in D(x_{i+1})}(s(v_0, w))$ for the minimum value $v_0 = \min(D(x_i))$ (respectively the maximum value) can be performed by scanning the minimum number of C-stretches of values in $D(x_{i+1})$. Dans le contexte de la contrainte INCREASING_NVALUE, qui restreint le nombre de valeurs distinctes affectées à une séquence de variables, de sorte que chaque variable de la séquence soit inférieure ou égale à la variable la succédant immédiatement. Formellement, elle se définie de la manière suivante (see [BEL 10b] for more details) :

Définition 6.2. INCREASING_NVALUE(N, X) *est définie par une variable* N *et une séquence de* n *variables* $X = [x_0, \ldots, x_{n-1}]$. *Soit une instantiation de* $[N, x_0, \ldots, x_{n-1}]$. *Elle satisfait* INCREASING_NVALUE(N, X) *ssi:*

1) N *est égale au nombre de valeurs distinctes affectées aux variables* X,

2) $\forall i \in [0, n-2]$, $x_i \leq x_{i+1}$.

En pratique, INCREASING_NVALUE(N, X) *se reformule par* SEQ_BIN$(N, X, =, \leq)$ *et vérifie la propriété de continuité du comptage, assurant ainsi un filtrage GAC.*

Assurer la GAC pour une contrainte NVALUE est un problème NP-Difficile, et les algorithmes de filtrage de cette contrainte sont connus pour être assez peu efficaces pour des domaines de variables énumérés [BES 05a]. Dans notre implémentation, nous utilisons une représentation de NVALUE qui est basée sur les contraintes d'occurrence de Choco. L'application choisie est basée un problème d'affectation de machine virtuelles dans les grappes de calcul. En quelques mots, *Entropy*[1] [HER 09] fournit un moteur autonome et flexible pour manipuler l'état et la position de VMs (machines virtuelles) sur les différents nœuds actifs composant une grappe. Ce moteur

1. http://entropy.gforge.inria.fr

est basé sur la programmation par contraintes [HER 11]. Il fournit un modèle central dédié à l'affectation de VMs à des nœuds, et permet de personnaliser cette affectation à l'aide de contraintes qui modélisent des besoins exprimés par des utilisateurs et des administrateurs.

Le modèle central définit chaque nœud (les ressources) par son CPU et sa capacité mémoire, et chaque VM (les tâches) par sa demande en CPU et en mémoire pour s'exécuter. La partie traitée via un modèle en programmation par contraintes consiste à calculer une affectation de chaque VM qui (i) satisfait la demande en ressources (CPU et mémoire) des VMs et (ii) utilise un nombre minimum de nœuds. Au final, la libération de nœuds peut faire que davantage de tâches seront acceptées dans la grappe, ou bien peut permettre d'éteindre les nœuds non utilisés pour économiser l'énergie. Dans ce problème, deux parties peuvent être distinguées : (1) l'affectation de nœuds en fonction de la capacité de ressources: il s'agit d'un problème de bin-packing 2D. Il est modélisé par un ensemble de contraintes *sac à dos* associées avec chaque nœud. La propagation est basée sur l'utilisation de la contrainte COSTREGULAR [DEM 06], afin de traiter simultanément les deux dimensions de ressource. (2) Réduction du nombre de nœuds utilisés pour instancier toutes les VMs. Les VMs sont classées en fonction de leur consommation en CPU et en mémoire (il existe donc naturellement des classes d'équivalence entre les VMs). Les contraintes NVALUE et INCREASING_NVALUE sont utilisées pour modéliser cette partie du problème. Nous évaluons alors l'effet de la contrainte INCREASING_NVALUE utilisée comme contrainte implicite sur des classes d'équivalence de machines virtuelles pour les lesquelles la contrainte INCREASING_NVALUE peut être utilisée pour casser des symétries. En pratique, les résultats obtenus par un modèle incluant des contraintes INCREASING_NVALUE montrent un gain modéré en terme de temps de résolution (3%), alors que le gain en nombre d'échecs est plus significatif (35% en moyenne).

6.4. Conclusion

La contribution de ce travail est une caractérisation fine d'une classe de contraintes de comptage pour laquelle nous avons pu mettre en avant les conditions d'un principe général de filtrage basé sur un algorithme atteignant l'arc-consistance généralisée. Nous avons aussi caractérisé les propriétés qui assurent qu'un tel algorithme puisse s'exécuter en temps linéaire sur la somme des tailles des domaines, à la fois en temps

et en espace. Les travaux résumés dans ce chapitre ont fait l'objet d'améliorations et d'extension par Katsirelos et al. [KAT 12].

La conclusion plus générale sur la portée d'un tel travail, incluant les conséquences pour l'avenir des contraintes globales, repose sur la nécessité de trouver un compromis entre le développement ad hoc d'un algorithme de filtrage dédié à un sous-problème particulier (avec pour seul objectif la performance en temps de calcul et/ou consommation mémoire) et une nécessaire prise de recul consistant à ne diffuser que des algorithmes de filtrages dont la portée (c.-à-d. l'utilisation pratique) a été montrée comme très générale et levant des verrous essentiels dans l'avancée de notre communauté. La démarche présentée ici se veut beaucoup plus pragmatique, dans le sens où la préoccupation première a été de généraliser un principe algorithmique, jusque là trop spécifique, en repartant des définitions et propriétés de base. Elle montre cependant la nécessité de prise de recul autour des outils algorithmiques que constituent les contraintes globales.

Chapitre 7

Approche probabiliste des contraintes globales

Sommaire

Qui n'a jamais observé dans un système de programmation par contraintes le fait qu'une contrainte globale semble "inutile" ? Par inutile, nous entendons le fait que la remplacer par un algorithme de filtrage plus faible (une décomposition, un algorithme assurant la consistance aux bornes lorsque l'algorithme initial assurait l'arc consistance généralisée) permet de converger plus rapidement vers une solution même si le nombre de points de choix et de retour-arrière augmente. Le plus bel exemple est la contrainte `allDifferent` ; en pratique, l'algorithme assurant la consistance d'arc généralisée est très rarement plus efficace que son pendant assurant la consistance aux bornes. Ceci a conduit beaucoup d'outils à "ruser" dans l'utilisation de cette contrainte en utilisant des mécanismes de choix automatique pour sélectionner l'algorithme à appliquer. Très peu de publications ont tenté de venir formaliser ces comportements [KAT 06], la raison tient simplement dans le fait que l'analyse en moyenne du comportement d'un algorithme de filtrage au sein d'un outil de programmation par contraintes est une tâche ardue, qui n'a aucune garantie d'être pertinente de par les hypothèses à effectuer sur la distribution des domaines des variables.

Cependant, dans le cadre de travaux que nous avons menés avec Jérémie Du Boisberranger, Danièle Gardy et Charlotte Truchet, nous avons tenté de déterminer, à partir de l'analyse en moyenne du comportement de la contrainte `allDifferent` pour son algorithme de consistance aux bornes, un oracle efficace prédisant sa capacité de filtrage [BOI 13]. Nous synthétisons dans la suite les résultats détaillés dans l'article et concluons sur les difficultés rencontrées, mais aussi sur les perspectives et le recul que ce travail nous a apportés.

7.1. Coût de la propagation et compromis de complexité

Au sein des systèmes de contraintes, les propagateurs représentent à eux seuls une part très significative du coût algorithmique de l'algorithme de propagation. Il est aisé d'observer que l'effet algorithmique (c.-à-d., la réduction des domaines de variables par filtrage) des propagateurs n'est pas uniformément distribué tout au long du processus de résolution. Autrement dit, ce type d'algorithme est très sensible aux conditions d'application, dans le sens où le processus de propagation les appelle de manière systématique sans garantie que leur application ait un effet en terme de réduction de domaines. Ce phénomène a été très peu exploré par la communauté de

programmation par contraintes qui a principalement concentrée ses efforts sur la complexité au pire cas des algorithmes de propagation [ROS 06]. Cependant, les travaux de Katriel [KAT 06] ont permis de mettre avant ce phénomène d'équilibre subtil entre performance (complexité en temps) et efficacité (réduction des domaines) dans les algorithmes de filtrage. Ces travaux, concentrés sur l'étude de la contrainte *Global Cardinality*, ont permis de montrer qu'il était possible de faire une analyse plus fine des conditions de filtrage afin de diminuer le nombre d'appel au propagateur concerné durant le processus de résolution, en particulier par l'observation des variables "importantes"dans la contrainte (c.-à-d., celles pour lesquelles la suppression de valeurs va entrainer une effet sur les autres variables). Bien entendu, déterminer ces conditions de filtrage a, là encore, coût algorithmique non négligeable et la conclusion des travaux menés a été d'observer qu'il était très difficile d'amortir ce dernier en pratique.

Bien sur, observer la complexité temporelle au pire cas d'un propagateur ne fournit pas assez d'information sur son efficacité durant le processus de résolution. Cependant, évaluer la complexité en moyenne est un travail autrement plus difficile et reste une voix jamais explorée par notre communauté. De plus, comme montré dans [KAT 06], le problème de suspendre des appels aux propagateurs (un moyen "simple"pour en mesurer l'efficacité) est très complexe pour deux raisons : tout d'abord, oublier une valeur inconsistante peut conduire à une explosion de la combinatoire dans la suite du processus de résolution, et par conséquent à un nombre beaucoup plus important d'appels aux propagateurs ; Et il faut de plus déterminer un indicateur d'efficacité qui puisse être calculé de manière très rapide (plus rapidement que l'application du filtrage lui-même) afin de s'assurer qu'il soit amorti dans le temps, c'est finalement un équilibre très complexe à trouver.

Ce chapitre synthétise don les résultats obtenus dans [BOI 13], pour le cas de l'étude théorique du comportement du propagateur de la contrainte ALLDIFFERENT dans sa version de consistance aux bornes. Nous avons montré comment, étant donné un ensemble de variables, nous pouvions déterminer un indicateur probabiliste pour la consistance aux bornes d'une contrainte ALLDIFFERENT. Nous avons ensuite montré comment une telle prédiction pouvait être asymptotiquement estimée en temps constant suivant certaines quantités relatives aux variables et aux domaines impliqués. les expérimentations ont montré que la précision étant suffisante pour une estimation pratique du comportment de l'algorithme. Le positionnement de ce travail est donc la

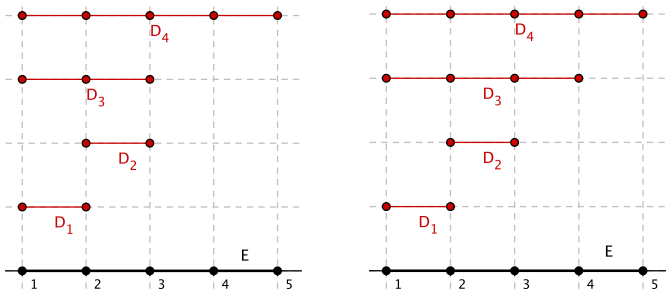
veine de [KAT 06] mais fournit un autre modèle pour approximer le comportement du filtrage à partir de probabilité de retrait d'une valeur d'au moins un domaine.

Le chapitre va s'organiser de la manière suivante : la prochaine section sera consacrée à un rappel de la définition de consistance aux bornes dans le cadre de la contrainte ALLDIFFERENT; Nous nous intéresserons ensuite à caractériser les situations pour lesquelles la contrainte reste sans effet et donc les situations pouvant conduire à ne pas propager cette dernière ; Enfin, nous présenterons notre modèle probabiliste de prédiction ; Finalement, les indicateurs exacts et asymptotique seront présentés et une première évaluation fournie ; Une conclusion sur les perspectives de ces travaux sera alors proposée.

7.2. Un modèle probabiliste pour la contrainte ALLDIFFERENT.

Nous avons choisi d'étudier la contrainte ALLDIFFERENT de part son impact sur la communauté et la variété des études qui ont été menées autour d'elle. Ceci a conduit à une très grande variété d'algorithmes de filtrage étudiés allant d'algorithmes assurant l'arc consistance globale (GAC) [RÉG 94] à des algorithmes assurant la consistance aux bornes (BC) [PUG 98]. Le lecture intéressé pourra se référer aux deux synthèses suivantes de Van Hoeve[HOE 01] ou Gent et al. [GEN 08] afin de synthétiser les différentes études autour de ces algorithmes.

Nous concentrons notre étude autour de la propriété de consistance aux bornes. Intuitivement, on dira qu'une contrainte sur n variables est consistante aux bornes si, en supposant que les domaines des variables concernées sont représentés par des intervalles, pour tout choix d'affectation d'une variable à la borne inférieure ou supérieure de son domaine, il est possible de trouver une affectation globale des $n-1$ variables restantes satisfaisant la contrainte. Nous nous intéressons ici à dériver cette définition afin d'introduire un modèle probabiliste pour la BC de la contrainte ALLDIFFERENT. Précisément, nous allons chercher à déterminer la probabilité pour une instanciation partielle des variables d'une contrainte ALLDIFFERENT satisfaisant la BC, de le rester après l'affectation d'une variable.



(a) Configuration inconsistante (b) Configuration consistante

Figure 7.1. *Deux situations pour la contrainte* ALLDIFFERENT

7.2.1. *Définitions et notations*

Considérons une contrainte ALLDIFFERENT sur n variables $V_1...V_n$, dont les domaines respectifs sont $D_1...D_n$ de taille d_i, $1 \leq i \leq n$. Nous supposons la taille de chaque domaine comme étant supérieure ou égal à 2, dans le cas contraire la variable correspondante est dite instanciée. Nous nous concentrons icic sur le cas de la consistance aux bornes [PUG 98] et nous supposons que les domaines D_i sont des intervalles d'entiers. Dans la suite, l'union de tous les domaines est noté $E = \bigcup_{1 \leq i \leq n} D_i$ et sa taille m. Notons que module un renommage des valeurs dans les D_i, leur union E sera aussi un intervalle d'entiers. Pour un ensemble I, nous écrirons $I \subset E$ pour dire que I est un sous-ensemble de E et que I est aussi un intervalle. Pour un intervalle $I \subset E$, nous noterons la borne inférieure (resp. la borne supérieure) de I par $\underline{I}$ (resp. $\overline{I}$). on aura donc $I = [\underline{I}...\overline{I}]$. Pour une valeur $x \in E$, $pos(x)$ sera la position de la valeur x dans E. Par convention, on dira que $pos(\underline{E}) = 1$.

Un exemple de configuration de domaines non consistante aux bornes pour une contrainte ALLDIFFERENT basé sur quatre variables, la figure 7.1(a) montre les domaines des variables $V_1...V_4$ respectivement définies sur $[1...2]$, $[2...3]$, $[1...3]$ et $[1...5]$. Il est clair que la valeur 1, qui est la borne inférieure de D_4, ne peut apparaitre dans une solution. La figure 7.1(b) est identique à la figure 7.1(a) modulo la modification

du domaine D_3 prenant ses valeurs dans $[1...4]$. Dans ce contexte, la configuration devient consistante aux bornes puisque toutes les valeurs extrêmes des domaines peuvent être utilisées dans au moins une solution.

Définition 7.1. *Soit $I \subset E$. Nous définissons K_I comme l'ensemble des variables pour lesquelles les domaines sont des sous-intervalles de I : $K_I = \{i$ tel que $D_i \subset I\}$.*

Par exemple, dans la figure 7.1(a), nous avons $K_{[1...3]} = \{1, 2, 3\}$, $K_{[3...5]} = \emptyset$ et $K_{[1...2]} = \{1\}$. Certains sous-intervalles de E jouent un rôle particulier puisqu'ils contiennent juste assez de valeurs pour assurer que chaque variable de K_I puisse être affectée à une valeur.[1] Par conséquent, les variables qui n'appartiennent pas à K_I ne peuvent pas prendre leurs valeurs dans I. Pour des raisons techniques, nous reformulons la définition classique de [HOE 01] dans un format équivalent :

Proposition 7.1. *Une contrainte* ALLDIFFERENT *sur un ensemble de variables $V_1...V_n$ de domaines $D_1...D_n$ est* consistante aux bornes *si et seulement si pour tout $I \subset E$, une des conditions suivantes est vérifiée :*

1) $|K_I| < |I|$,

2) $|K_I| = |I|$ et $\forall i \notin K_I, I \cap \{\underline{D_i}, \overline{D_i}\} = \emptyset$.

Cette propriété est utile pour déterminer la BC de la contrainte ALLDIFFERENT de manière globale, au début du processus de résolution par exemple. En pratique, le maintient du niveau de consistance se fait de manière incrémentale : on ne réagit qu'aux modifications des domaines dans le réseau. La question principale peut alors se poser de la manière suivante :

> *Sachant qu'une contrainte* ALLDIFFERENT *est initialement BC, sous quelles conditions peut-on assurer qu'elle reste BC après affectation de l'une de ses variables ?*

1. Les sous-intervalles I de E tels que $|K_I| = |I|$ sont appelés *intervalles de Hall*. Ils apparaissent fréquemment dans les caractérisations de la consistance aux bornes de la contrainte ALLDIFFERENT.

7.2.2. *Consistance aux bornes après l'affectation d'une variable*

Nous nous intéressons ici à l'effet de l'affectation d'une variable sur les domaines et sur la propriété de consistance aux bornes pour la contrainte ALLDIFFERENT. Modulo un renomage des variables, on peut supposer sans pertes de généralités que l'affectation est faite sur la variable V_n, qui est affectée à la valeur $x \in D_n$. Nous supposons de plus que la valeur x a été supprimée des domaines de toutes les autres variables lorsque nécessaire.

Suite à une affectation, on peut décrire la situation suivante : le domaine D_n a disparu (dans le sens où il est réduit à un singleton $\{x\}$, et pour $i \neq n$, deux cas peuvent survenir. Soit $x \notin D_i$ alors, le domaine reste inchangé, soit $x \in D_i$ alors, le domaine D_i est maintenant l'union de deux intervalles disjoints $[\underline{D_i}...x-1]$ and $[x+1...\overline{D_i}]$. La question du maintient de la consistance aux bornes ne parait alors plus vraiment pertinente par le simple fait que certains domaines ne sont plus des intervalles. Néanmoins, nous pouvons définir de nouveaux domaines $D_i^{'}$=$D_i \setminus \{x\}$, qui ne sont pas des sous-intervalles de E mais de $E^{'} := [\underline{E}...x-1] \cup [x+1...\overline{E}]$ puisque toutes les valeurs de $E^{'}$ entre $\underline{D_i}$ et $\overline{D_i}$ sont dans $D_i^{'}$. Nous obtenons alors la définition suivante :

Définition 7.2. *La contrainte* ALLDIFFERENT reste BC *après l'affectation de V_n si et seulement si la contrainte* ALLDIFFERENT *est BC par rapport aux nouveaux domaines $D_1^{'}...D_{n-1}^{'}$.*

De plus, pour tout sous-intervalle $I^{'} \subset E^{'}$, nous pouvons définir un intervalle associé $I \subset E$ tel que :

$$I = \begin{cases} I^{'} \cup \{x\} \text{ si I est un sous} - \text{intervalle de } E; \\ I \text{ sinon.} \end{cases}$$

La proposition suivante détaille dans quels cas la contrainte, vérifiant la BC sur $V_1...V_n$, *reste* BC (comme défini ci-dessus) après l'affectation de la variable V_n. Ce résultat, parfaitement classique pour la communauté, est ici rappelé afin de clarifier les faits.

Proposition 7.2. *Étant donné les notations précédentes, la contrainte* ALLDIFFERENT *reste BC après l'affectation d'une variable V_n à une valeur x si et seulement si pour tout $I^{'} \subset E^{'}$, tel que $I = I^{'} \cup \{x\}$ et $D_n \not\subset I$, aucune des deux assertions suivantes n'est vérifiée :*

(i) $|K_I| = |I|$,

(ii) $|K_I| = |I| - 1$ *et il existe* $i \notin K_I$ *tel que* $\underline{D_i} \in I$ *ou* $\overline{D_i} \in I$.

7.3. Probabilité de conserver la consistance aux bornes

Finalement, déterminer la consistance aux bornes de la contrainte ALLDIFFERENT revient à étudier la taille et la position relative des domaines les uns par rapporta aux autres. Cependant, on peut noter qu'il n'est pas toujours nécessaire de connaitre précisément la forme de chaque domaine pour décider de la consistance de la contrainte. Par exemple, une contrainte ALLDIFFERENT sur trois variables de domaines de taille 3 est toujours BC, quelque soit la position exacte des domaines sur un intervalle donné. Si les domaines sont de taille 2 alors il devient évident que la consistance aux bornes devient dépendante de la position précise des domaines, à moins que l'union E des domaines soit elle aussi de taille 2, auquel cas la contrainte sera toujours inconsistante.

Ces remarques relativement triviales montrent qu'une connaissance, même partielle, sur l'agencement relatif des domaines est parfois suffisante pour déterminer certaines propriétés sur la consistance. Ces éléments constituent la base de notre approche probabiliste : nous supposerons que nous connaissons à tout instant l'état de l'union des domaines E (qui est assez facile à observer en pratique bien que cela puisse avoir un coût), mais les domaines eux-même deviendront des variables discrètes aléatoires (au sens probabiliste cette fois-ci) respectant une loi de distribution uniforme sur l'ensemble $\mathcal{I}(E)$ des sous-intervalles de E, de taille ≥ 2 : Les domaines sont donc quelque part "perdus", dans le sens où ils ont été probabilisés. Leur taille et position exactes dans E deviennent des inconnues ; seulement quelques quantités macroscopiques peuvent être utilisées.

Cette section va résumer les résultats relatifs à l'évaluation de la probabilité, pour une contrainte ALLDIFFERENT, de rester BC après l'affectation de l'une de ses variables, sous les hypothèses suivantes :

– Pour $E = \bigcup_{i=1}^{n} D_i$, nous supposons :

A1. *E est un intervalle d'entiers connu de taille m; sans perte de généralités, nous prendrons : $E = [1...m]$.*

– Les domaines $D_1, ..., D_n$ sont modélisés par des variables aléatoires $\mathcal{D}_1, ..., \mathcal{D}_n$ de la manière suivante :

A2. *les variables $\mathcal{D}_i$ sont indépendantes et uniformément distribuées sur $\mathcal{I}(E) = \{[a...b], 1 \leq a < b \leq m\}$.*

À partir de l'hypothèse **A2**, l'échantillon $\mathcal{I}(E)$ est de taille $m(m-1)/2$ (les domaines de taille 1 étant interdits). Pour $J \subset E$ et $1 \leq i \leq n$, nous avons donc $P[D_i = J] = 2/m(m-1)$. En effet, il y a $m-1$ sous-intervalles de taille 2 possibles, $m-2$ de taille 3, ..., 1 de taille m.

Nous proposons maintenant une formule générale pour cette probabilité, ensuite, nous calculerons sa valeur asymptotique dans le cas où les variables observables sont assez grandes ; Nous montrerons aussi que cette approximation asymptotique est calculable en temps constant.

7.3.1. *Un calcul exact et approximation asymptotique*

Nous considérons d'abord une série de probabilités intermédiaires nécessaires pour écrire la probabilité générale de rester BC après affectation pour une contrainte ALLDIFFERENT.

Proposition 7.3. *Pour un intervalle $I \subset E$ et un domaine $\mathcal{D}$ défini avec une distribution uniforme sur $\mathcal{I}(E)$, avec $m = |E|$ et $l = |I|$, soit p_l et q_l les probabilités respectives que $\mathcal{D} \subset I$, et que soit $\mathcal{D} \cap I = \emptyset$ ou $\underline{\mathcal{D}} < \underline{I} < \overline{I} < \overline{\mathcal{D}}$. Alors*

$$p_l = \frac{l(l-1)}{m(m-1)}; \qquad q_l = \frac{(m-l)(m-l-1)}{m(m-1)}.$$

Afin de pouvoir calculer la probabilité que la contrainte reste BC, nous pouvons maintenant injecter dans la Proposition 7.2 les quantités précédentes, ce qui conduit au théorème suivant :

Théorème 7.1. *Étant donné une contrainte* ALLDIFFERENT *sur les variables $V_1, ..., V_n$, initialement BC. Supposons que E et les domaines $\mathcal{D}_i$, $1 \leq i < n$, satisfassent les hypothèses* **A1** *and* **A2**. *De plus, supposons que nous connaissons le domaine $D_n = [a...b]$. Alors la probabilité $P_{m,n,x,a,b}$ que la contrainte reste BC après l'affectation de la variable V_n à une valeur x est*

$$P_{m,n,x,a,b} = \prod_{l=1}^{n-2} (1 - P^{(1)}_{m,n,l} - P^{(2)}_{m,n,l})^{\Phi(m,l,x,a,b)}$$

où $\Phi(m, l, x, a, b)$ *est défini par*

$$\min(x, m-l) - \max(1, x-l) + 1$$

Si $l < b - a$ *et comme*

$$\min(x, m-l) - \max(1, x-l) - \min(a, m-l) + \max(1, b-l)$$

sinon, et où

$$\begin{aligned} P^{(1)}_{m,n,l} &= \binom{n-1}{l+1} p_{l+1}^{l+1} (1 - p_{l+1})^{n-l-2}, \\ P^{(2)}_{m,n,l} &= \binom{n-1}{l} p_{l+1}^{l} \left((1 - p_{l+1})^{n-l-1} - q_{l+1}^{n-l-1} \right), \end{aligned}$$

avec p_l *et* q_l *donnés dans la Proposition 7.3.*

À partir de l'expression de $P_{m,n,x,a,b}$ donnée par le théorème 7.1, il est possible de calculer la probabilité $P_{m,n,x,a,b}$ avec une complexité $O(n)$ (avec une constante importante). Cependant, utiliser un tel indicateur probabiliste pour la consistance aux bornes dans un outil de programmation par contraintes nécessite de prendre en considération le fait qu'il va devoir être évalué de manière répétitive (au fil des étapes de l'algorithme de propagation). C'est un surcoût non négligeable qui devra s'avérer rentable même pour de grandes valeurs de n et m. Ainsi, la formule du théorème 7.1 ne peut être utilisée telle quelle et nécessite une approximation à la fois précise et calculable "rapidement". La proposition suivante propose donc une approximation asymptotique dans la taille des puissances de $1/m$. Deux comportements distincts ont été isolés. Sans surprise, ces comportements dépendent de la distribution de n par rapport à m. Dans le premier cas, n est proportionnel à m, ce qui correspond à un contrainte ALLDIFFERENT avec beaucoup de valeurs et peu de variables. Le second cas, $m - n = o(m)$, correspond à une contrainte ALLDIFFERENT pour laquelle il y a à peu près autant de variables que de valeurs dans les domaines.

Théorème 7.2. *Étant donné une contrainte* ALLDIFFERENT *sur les variables*

$V_1, ..., V_n$ Supposons que les domaines $\mathcal{D}_1, ..., \mathcal{D}_{n-1}$ suivent une distribution uniforme sur E. Soit $D_n = [a...b]$. Définissons une fonction $\Psi(m, x, a, b)$ pour $1 \leq a \leq x \leq b \leq m$ et $a \neq b$ par

– $\Psi(m, x, a, a+1) = 1$, sauf pour $\Psi(m, 1, 1, 2) = \Psi(m, m, m-1, m) = 0$;

– Si $b > a+1$ alors $\Psi(m, x, a, b) = 2$, sauf pour $\Psi(m, 1, 1, b) = \Psi(m, m, a, m) = 1$.

Alors, la probabilité $P_{m,n,x,a,b}$ que la contrainte reste BC a que la contrainte reste BC après l'affectation de la variable V_n à une valeur x a la valeur asymptotique suivante

– Si $n = \rho m,\ \rho < 1$,

$$1 - \Psi(m, x, a, b)\frac{2\rho(1 - e^{-4\rho})}{m} + O\left(\frac{1}{m^2}\right);$$

– Si $n = m - i,\ i = o(m)$,

$$e^{C_i}\left(1 - \Psi(m, x, a, b)\frac{2(1 - e^{-4}) + D_i}{m} + O\left(\frac{1}{m^2}\right)\right).$$

Lorsque $n = m - i$ avec $i = o(m)$, les constantes C_i et D_i (qui dépendent aussi de x et a) peuvent être exprimées comme

$$\sum_{a-i \leq j < x-i} (j + i + 1 - a)\varepsilon_{i,j} + (x - a) \sum_{j \geq x-i} \varepsilon_{i,j},$$

avec $\varepsilon_{i,j}$ égal à $\log(1 - f_{i,j})$ pour C_i et $g_{i,j}/(1 - f_{i,j})$ pour D_i,

$$f_{i,j} = \lambda_{i,j}\left(1 + \frac{j}{2(i + j)}\right)$$

et

$$g_{i,j} = \lambda_{i,j}\left(i(i+1) + \frac{j}{4}(3i + 5) + \frac{j(i^2 - 1)}{4(i + j)}\right)$$

avec

$$\lambda(i, j) = \frac{(i + j)^j 2^j e^{-2(i+j)}}{j!}.$$

Le théorème 7.2 est important pour deux raisons. Tout d'abord, il fournit un indicateur probabiliste rapide à calculer (cf. discussion de la section 7.3.2). Ensuite, il met

en lumière deux régime de fonctionnement pour l'algorithme de la contrainte ALLDIFFERENT. Il formalise une preuve rigoureuse d'un comportement longtemps observé dans le folklore de la CP : la report entre n et m dans une contrainte ALLDIFFERENT est l'ingrédient clé pour déterminer la qualité du filtrage que l'on peut espérer. L'expression proposée par le théorème 7.2 a mis en avant le fait que la probabilité de rester consistant aux bornes avait une limite asymptotique égale à 1 dans le premier cas et strictement plus petite que 1 dans le second. Ainsi, pour de grandes valeurs de m, propager ALLDIFFERENT de manière systématique est le plus souvent inutile, modulo le cas $m - n = o(m)$, c.-à-d. lorsque m et n sont très proches.

7.3.2. *Calcul pratique d'un indicateur*

Nous venons de voir que le théorème 7.2 fournit une approximation asymptotique (quand m devient grand) de la probabilité de rester consistant aux bornes pour une contrainte ALLDIFFERENT.

Lorsque $n = \rho\, m$, nous obtenons une solution en forme close pour l'approximation, qui peut être calculée en temps constant (les constantes n'étant cependant pas négligeables en pratique). Dans le cas où $n = m - i$, les constantes C_i et D_i, bien qu'indépendantes de m et n (elles dépendant de a et x), n'ont pas de solution en forme close. Cependant, elles sont données au travers de leurs limites sur des sommes infinies. Néanmoins, une bonne approximation peut être obtenue avec un nombre fini de termes. En effet, les termes $f_{i,j}$ et $g_{i,j}$ sont exponentiellement petit pour des tailles de i et j tendant vers $+\infty$. Par exemple, la valeur $\log(1 - f_{i,j})$ est à peu près de l'ordre de l'exponentielle $(2/e)^j$. Ainsi, la convergence vers la limite est assez rapide pour envisager un calcul efficace. Une autre question pratique consiste à choisir parmi les deux régimes de fonctionnement de la formule. En d'autres termes, il faut choisir quand est-ce que n est suffisamment proche de m ? Le théorème 7.2 est valide pour $m \to \infty$ mais, se divise en deux cas selon que n reste proche d'une constante, ou $m - n = o(m)$.

Une exemple est donné dans la figure 7.2. On y retrouve la probabilité théorique (trait plein) et son approximation (en pointillés) pour des valeurs de $m = 25$ ou $m = 50$, et pour une ratio variant sur n/m. Dans les deux figures, les valeurs x, a et b sont arbitrairement fixées à respectivement 15, 3 et 9 La courbe en pointillés

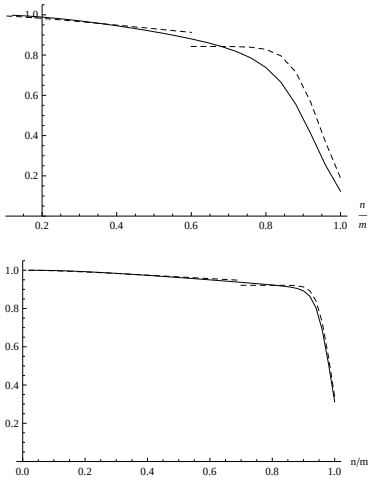


Figure 7.2. *Évaluation numérique des probabilités théorique (en trait plein) et approximée (en pointillé) de* $P_{25,n,15,3,19}$ *pour* $m = 25$ *(courbe du haut) ou* 50 *(courbe du bas), et pour* n *variant de 1 à* m*.*

présente une discontinuité sur un point. Ceci est dû au changement de régime : on applique le cas 1 pour $n \leq m - 2\sqrt{m}$ et le cas 2 pour $n > m - 2\sqrt{m}$, ce qui semble être un bon compromis. Remarquons aussi que même si $m = 50$ n'est certainement pas une grande valeur, l'approximation semble toujours plutôt bien coller aux valeurs attendues.

En pratique, les évaluations numériques faisant varier les valeurs de m et n ont indiqué que le meilleur compromis est de placer un seuil à $n = m - 2\sqrt{m}$ pour distinguer les deux cas du théorème 7.2. Notons aussi que ce seuil peut être utilisé comme une définition quantitative de l'efficience de la contrainte (va-t-elle avoir une effet ?). Ceci conduit donc à la définition suivante :

Définition 7.3. *Une contrainte* ALLDIFFERENT *sur* n *variables et* m *valeurs est* bien dimensionnée *si et seulement si* $m - n < 2\sqrt{m}$*.*

7.4. Conclusion et poursuite du travail

Ce chapitre a proposé une étude probabiliste du comportement de l'algorithme de filtrage assurant la consistance aux bornes de la contrainte ALLDIFFERENT. Cette

étude est un premier pas vers des stratégies d'adaptation automatique du comportement des algorithmes de filtrage. Ainsi, Nous avons été en mesure de calculer la probabilité de rester consistant aux bornes pour une contrainte ALLDIFFERENT. Le défit d'une telle approche reste de proposer un traitement algorithmique qui puisse être amorti durant le processus de résolution : une approximation asymptotique de la formule proposée a permis de mettre en avant un calcul en temps constant.

Cependant, comme tout modèle conceptuel, le nôtre repose sur une simplification de la réalité exprimée par les hypothèses mathématiques qui ont été utilisés (distribution uniforme et indépendance de l'état des domaines des variables). Nous avons ainsi supposé que les domaines étaient indépendants et identiquement distribués par des variables aléatoires. Tant que nous nous limitons à une seule contrainte ALLDIFFERENT, ces hypothèses semblent raisonnables. Bien sur le choix d'une distribution uniforme reste une choix de modélisation fort qui doit être discuté. Cependant, une partie de nos travaux n'ont pas été rapportés ici [BOI 11]. En particulier, nous avons étudié d'autres formes de distributions (en particulier celle fixant la taille de tous les domaines à une unique valeur) classiques dans les problèmes académiques. Les résultats pratiques diffèrent très peu et ils nous semblent que ces deux cas couvrent déjà une large majorité des situations rencontrées en pratique. L'intégration dans les systèmes de contraintes actuels semble cependant beaucoup plus délicate. Tout d'abord, le théorème 7.2 est valide dans une limite $m \to \infty$. En pratique, cela signifie que m doit être relativement grand. Nos expérimentations montrent que la zone de validité se situe donc plutôt au début du processus de résolution. Ensuite, une question importante reste la caractérisation d'un seuil pour la valeur de la probabilité de rester consistant aux bornes. Enfin, un point important et récurrent à toute optimisation "théorique"est son intégration dans un système de programmation par contraintes. Pour cela, il faut d'abord considérer la forme de la contrainte elle-même et son interaction avec le moteur de propagation : présence ou non d'une gestion évènementielle du filtrage, ordre de révision de la propagation guidé par la priorité des propagateurs, etc.

Dans ce contexte, les premiers résultats montrent un impact significatif sur le processus de résolution. Cependant, le niveau d'optimisation de l'algorithme de filtrage de la contrainte ALLDIFFERENT ainsi que du schéma de propagation global semblent avoir un impact réel sur les résultats obtenus. Ces premiers résultats nous poussent

donc à une analyse plus fine du comportement des algorithmes de filtrage au sein des schémas de propagation, et nous montrent l'importance de l'analyse en moyenne de ce mécanisme afin de déterminer automatiquement le pertinence d'un algorithme de filtrage *fort*.

Chapitre 8

Conclusion et perspectives générales

8.1. Conclusion

En conclusion, les travaux synthétisés dans ce manuscrit ont montré, si cela était encore nécessaire, que la construction d'un outil de programmation par contraintes n'est pas une tâche aisée ; tant de par la difficulté de suivre et d'intégrer de manière uniforme l'état de l'art (autour du triptyque propagation, recherche, filtrage) que de rester dans le cadre d'un outil générique et utilisable de manière déclarative. Tout ceci dans le contexte d'un délicat équilibre entre l'outil "boîte noire" et la bibliothèque nécessitant des compétences pour composer les briques formant l'outil de résolution.

Sur la partie propagation, nos travaux viennent compléter un vide au niveau de la conception flexible de moteurs de propagation. Que l'on juge ce travail inutile pour l'amélioration des performances des moteurs de propagation est une chose, mais il constitue une réelle avancée dans le prototypage aidant au développement de moteurs de propagation dédiés à un profil de problème. De manière plus pragmatique, la communauté, ayant fait le deuil (temporaire) d'un paradigme purement déclaratif, accepte de devoir spécifier sa propre heuristique de recherche au profit de performances accrues; pourquoi serait-elle réfractaire à une idée similaire concernant la propagation, qui constitue tout de même le cœur de la programmation par contraintes ?

Sur la partie recherche, notre démarche est partie du constat que les stratégies et heuristiques de recherche complètes ont atteint un seuil qui semble aujourd'hui infranchissable sans apport de nouveaux concepts (cf. travaux sur les explications, hybridation SAT, etc.). De plus, dans le cadre des problèmes d'optimisation, les considérations opérationnelles, remontées par les industriels, sont souvent décorrélées de toute notion d'optimalité (on souhaite une "bonne"solution obtenue dans un temps raisonnable). Dans ce cadre, intégrer la démarche "intelligente" des explications au sein de stratégies d'optimisation incomplètes comme le LNS, nous est apparu comme pertinent. Nous répondons ainsi à une demande de mise en place rapide de stratégies de recherche incomplètes mais adaptative qui ne nécessitent pas de développement spécifique, dans un premier temps. Nous prenons donc quelque part un contre-pied de la partie propagation, où plutôt que de donner plus de liberté à l'utilisateur, nous lui proposons de laisser un système "intelligent" configurer sa stratégie de recherche.

Sur la partie filtrage qui constitue le "nerf de la guerre" dans les systèmes de contraintes, nous avons mis l'accent sur l'équilibre à trouver entre algorithme de filtrages

dédiés et nécessité de capitaliser ces derniers afin de limiter l'introduction de concepts algorithmiques souvent très subtils.

Aujourd'hui, il fait consensus que les contraintes sont définies en intention dans les systèmes et qu'au moins un algorithme de filtrage est alors associé à chacune. Cette nécessité est d'autant plus vraie dans le cas de graphes qui constituent des structures de données très complexes à manipuler et pourtant souvent présentes dans les grands familles de problèmes combinatoires.

Pourtant, ce caractère dédié des algorithmes de filtrage ne doit pas occulter la nécessité de capitaliser par tous les moyens ces derniers dans la mesure du possible et de déterminer des compromis généricité / performance pour chacun. C'est dans cette démarche que nous avons proposé les travaux autour des contraintes de comptage sur des séquences de variables. Bien que modeste, ce travail met en avant cette démarche de capitalisation de la connaissance et de montée en abstraction des algorithmes de filtrages initialement dédiés à un problème particulier.

Enfin, notre dernière contribution re pose sur l'observation que le comportement des algorithmes de filtrage est encore aujourd'hui mal cerné, en particulier dans le cas des contraintes globales. L'analyse en moyenne (bien que très délicate et peu commune dans notre communauté) semble essentielle pour faire une utilisation raisonnable et raisonnée des contraintes globales dans l'objectif de pouvoir proposer un système capable d'adapter dynamiquement le comportement de ces algorithmes de filtrage au profil combinatoire du réseau de contraintes sous-jacent. Notre première étude autour du comportement de la contrainte classique ALLDIFFERENT ouvre de nombreuses perspectives mais pose aussi de nombreuses questions sur l'énergie à déployer pour obtenir de tels résultats.

8.2. Perspectives

L'évolution des systèmes de programmation par contraintes passe aujourd'hui par une analyse plus fine de leur comportement à la fois sur le volet pratique et théorique. En ce sens il me semble aujourd'hui fondamental de se donner deux axes de poursuites dans l'étude théorique des outils :

– Analyse théorique des systèmes au sens de la complexité en moyenne des algorithmes de filtrage (internes auxcontraintes) mais aussi des schémas de propagation

et des stratégies de recherche. Les analyses au pire cas fournies aujourd'hui par la communauté ne sont plus suffisantes pour comprendre les différences de comportements entre modèles pour un même problème, et encore plus entre stratégies de résolution (propagation + recherche) pour un même modèle d'un problème. Cette analyse sera exigeante et nécessitera d'accepter de revenir sur des systèmes simples et dénués de toute optimisation pratique afin de comprendre les comportements de nos outils, mais elle permettra de définir des critères rationnels et justifiés pour optimiser le comportement de certains algorithmes dans les systèmes à base de contraintes (nécessité d'atteindre un point fixe, choix d'un niveau de filtrage pour une même contrainte, adaptation automatique de l'heuristique).

– Un complément indissociable du point précédent consiste à mener une analyse statistique des systèmes et des modèles. On retrouve aujourd'hui cette notion d'analyse statistique au niveau des heuristiques de recherche [REF 04, MIC 12, PES 12] et beaucoup moins au niveau de la propagation, à l'exception de travaux très récents [LEC 11, BAL 13]. Elle est aussi de plus en plus présente dans l'aide à la modélisation avec des systèmes d'apprentissages de plus en plus évolués et convaincants au niveau opérationnel [BEL 12, BES 13]. Ces analyses du comportement "pratique"des systèmes permettent aujourd'hui de se rapprocher d'un système totalement autonome à même de résoudre seul un problème combinatoire à partir d'un modèle "simple"et/ou d'exemples caractérisant une solution et/ou une non-solution.

Ces perspectives théoriques ne doivent en aucun cas occulter les besoins et attentes de l'industrie vis-à-vis de la technologie des contraintes. Je ne retiens ici qu'une perspective qui me semble essentielle car devenue récurrente dans les applications métiers liées à l'optimisation : la prise en compte du caractère *opérationnel* des problèmes combinatoires (qu'ils soient en satisfaction ou en optimisation). Par opérationnel, j'entends que les demandes récurrentes de l'industrie nécessitent la prise en compte d'une dimension à la fois historique, mais aussi réactive aux aléas, dans la production d'une nouvelle solution.

Un exemple typique que j'ai pu rencontrer à ce jour concerne la conception d'emploi du temps pour des personnels. Aujourd'hui, de nombreux outils (commerciaux et ad hoc), basés sur différentes techniques issues de la recherche opérationnelle (méta-heuristiques, génération de colonnes, décomposition de Benders, etc.), proposent des

solutions totalement satisfaisantes, en termes d'optimisation d'un objectif, à une dimension stratégique du problème (par exemple pour dimensionner le personnel nécessaire pour satisfaire les contraintes). Mais lors de la mise en place de l'emploi du temps opérationnel (tâche quasi-journalière déterminant "quelle personne à quel poste"), il faut prendre en compte de nombreux aléas intervenus depuis la phase stratégique (arrêt maladie, préférences, vacances à solder, etc). Cette dimension opérationnelle nécessite une capacité d'interaction beaucoup plus importante dans le sens où l'on ne peut se permettre de répondre en quelques heures aux modifications demandées, ou que l'on doit être en mesure de prendre en compte une variété plus large de contraintes qui n'existaient pas au niveau stratégique. C'est à ce niveau que la programmation par contrainte doit prouver sa flexibilité et son efficacité en validant le fait qu'elle offre un panel de stratégies de recherche (basées sur la réparation de voisinage - LNS) et de propagation (à même de supporter l'injection et la suppression dynamique de contraintes ou de valeurs dans les domaines des variables). Alors l'enjeu n'est plus le calcul d'un nouvel optimum mais plutôt la réparation d'une solution existante sans dégradation excessive de l'objectif.

Ce type de problématique représente, à mon sens, un champ d'application très pertinent pour la programmation par contraintes puisqu'il nécessite de maîtriser tous les aspects de cette dernière et, prouve, s'il le fallait encore, qu'elle est à même d'attaquer de vrai problèmes industriels pour peu que l'on sache lever les derniers verrous freinant la totale flexibilité de ce paradigme.

Chapitre 9

Curriculum Vitae Scientifique

Xavier Lorca
né le 15 juillet 1981

7 Av. Toutes Joies, 44000 Nantes
Tél. : +33 (0)6 81 82 60 13

PhD - Maître Assistant

Équipe TASC – Mines Nantes – Département Informatique
LINA - INRIA - CNRS UMR 6241
4 rue Alfred Kastler, BP 20722, F-44307 Nantes Cedex 03
Tél.: +33 (0)2 51 85 83 48, Fax: +33 (0)2 51 85 82 49
xavier.lorca@mines-nantes.fr

9.1. Parcours professionnel

– **Juin. 2013 - auj. :** Animateur thème "Contraintes et Optimisation" – TASC et OPTI – LINA. En charge de l'animation scientifique des deux équipes autour des problématiques mixant optimisation discrète et globale. L'objectif est de renforcer la cohésion du thème, de contribuer à son identification au sein du LINA et de la fédération AtlanSTIC, et sa visibilité/rayonnement à l'extérieur du laboratoire.

– **Sept. 2011 - auj. :** Responsable de l'option "Génie Informatique pour l'Aide à la Décision" – Mines Nantes. La formation couvre 50% du cursus des élèves ingénieur entrant à l'école des Mines, soit 1,5 ans pour 630h EDT et 9 mois de stages.

– **Sept. 2009 - Oct. 2011 :** Responsable de la gestion et du budget matériel informatique Département Informatique – Mines Nantes. En charge de la gestion du budget relatif à l'investissement en matériel informatique pour le département. Mes attributions allaient de la négociation budgétaire avec la direction pour l'année avenir, à la gestion des commandes pour le personnel du département.

– **Oct. 2008 - auj. :** Maître Assistant en Informatique – Mines Nantes. Le projet de mon recrutement reposait sur l'évolution de l'outil Choco au sein de l'équipe TASC. L'objectif était de travailler à la fois sur le cœur de l'outil et sa stabilisation, mais aussi sur l'intégration des dernières évolutions en matière de stratégies de recherche et de propagation. Enfin se donner la capacité de communiquer largement (tant au niveau académique qu'industriel) autour des nouvelles capacités de l'outil.

– **Nov. 2007 - Sept. 2008 :** Ingénieur de Recherche, Projet Européen Net-WMS. Etude et analyse d'un modèle de programmation par contraintes couplant tournées de véhicules et contrainte de chargement (placement géométrique) de camions. L'objectif était d'analyser de manière pratique les interactions entre les deux parties du modèle afin de déterminer une stratégie de résolution adaptée.

9.2. Parcours universitaire

– **Oct. 2004 - Oct. 2007 :** Doctorat Informatique de l'Université de Nantes – *Contraintes de partitionnement de graphe.*

- Directeur : N. Beldiceanu.
- Rapporteurs : A. Bockmayr, G. Pesant.
- Examinateurs : G. F. Italiano, J.-X. Rampon , J.-C. Régin.

– **2003 - 2004** Diplôme d'Études Approfondies, Informatique, Université Montpellier II – LIRMM CNRS UMR 5506.

– **1999 - 2003 :** Maîtrise d'informatique, option intelligence artificielle – Université Montpellier II.

9.3. Valorisation et projets

– **ANR Laboratoire Commun TransOP (2014 - 2017) :** coordinateur et responsable d'un Laboratoire commun avec la société EuroDécision autour des problèmes de régulation opérationnelle. L'objet du laboratoire commun est de proposer un formalisme et des méthodes de résolution à même de réparer, en opération, les planifications tactiques pour aider le planificateur à répondre efficacement aux besoins de régulation (élaboration et suivi opérationnel des emplois du temps du personnel). [**montant :** 300k€]

– **MiniZinc Challenge (2013) :** Outil Choco médaillé d'argent dans deux catégories. Compétition annuelle autour des outils issus de la programmation par contraintes et de plus généralement de la recherche opérationnelle, regroupant environ 20 outils. Organisée en parallèle de la conférence "Principle and Practice of Constraint Programming"(CP).

– **ANR Infra-JVM (2012 - 2015) :** coordinateur local du projet. Conception d'une machine virtuelle java dédiée aux problématiques des outils de programmation par contraintes. En partenariat avec l'équipe REGAL du LIP6, nous étudions les possibilité d'utilisation de la plateforme VMKIT pour développer une machine virtuelle Java dédiée à l'outil Choco afin d'obtenir une gestion plus fine de la mémoire. [**montant :** 88,2k€]

– **Google grants (2012 - 2013) :** coordinateur avec Jean-Charles Régin (Pr., Université de Nice). *A Constraint Programming Based Traveling Salesman Problem Solver*. Google Focused Grant Program on Mathematical Optimization and Combinatorial Optimization in Europe. [**montant :** 50k€]

– **FCI-System (2010) :** formation à l'outil CHOCO – Planification de techniciens EDF-GDF. Dans le cadre de l'animation et de la diffusion autour de l'outil Choco, une semaine de formation a été dispensée au sein de la société FCI-System afin d'assurer la prise en main de l'outil. Cette

semaine de formation a été suivie de plusieurs jours de consulting autour d'un modèle Choco résolvant des problèmes d'organisation de tournées de techniciens en intervention pour EDF et GDF. [**montant :** 5k€]

– **LigéRO (2009 - 2013) :** coordinateur local du projet. Projet régional soutenant un groupe de recherche fédérant les acteurs du domaine de la recherche opérationnelle en région Pays de la Loire. Les actions ont consisté dans le soutient de groupes de travail, financement d'une thèse et soutient financier à l'organisation d'évènement de type ateliers ou conférences. [**montant (TASC) :** 6k€]

– **SAGEM (2007-2008) :** développement d'une solution pour la coordination du déplacement d'unités tactiques en milieu hostile (porté par N. Beldiceanu). L'objectif était de proposer une solution flexible embarquant les règles standard de déplacement et d'engagement au niveau militaire. L'approche proposée a consisté à modéliser le problème sous forme de graphes et de proposer une approche intégrée utilisant mes travaux de thèse. Les résultats ont montrés que les règles d'engagement (métier) sont tellement forte que le problème ne relève plus vraiment de l'optimisation et qu'un contexte de satisfaction est largement suffisant. [**montant :** 15k€]

9.4. Animation scientifique

– **Animateur du thème "Contraintes et Optimisation" – LINA.** Renforcer la cohésion du thème, contribuer à son identification au sein du LINA et de la fédération AtlanSTIC, et sa visibilité/rayonnement à l'extérieur du laboratoire. Les attributions : gestion des ressources provenant de la direction de LINA à la sélection des sujets de thèses remontés au niveau de l'école doctorale STIM.

– **Comités de programme de conférences :**

- Internationales : IJCAI'13 (membre senior P. Flener) ;
- Nationales : JFPC'13, JFPC'12, JFPC'11.

– **Relectures d'articles :**

- Journaux Internationaux : Constraints (6 art.), RAIRO (4 art.) ;
- Conférences internationales : CPAIOR'12 & 08 (4 art.), AAAI'06 (1 art.) et ECAI'06 (1 art.) ;
- Conférences nationales : JFPC'10 & 06 (7 art.), ROADEF'05 (1 art.).

– **Jury de thèse :** examinateur de la thèse de G. Richaud (2009). *Problèmes dynamiques et problèmes de dynamicité*.

– **Encadrement de thèses :**

- C. Prud'homme (2011 - 2014) : *Langages dédiés pour le pilotage de solveurs de contraintes*. Co-encadrement avec N. Jussien et R. Douence.

- J.G. Fages (2011 - auj.) : *Modélisation de graphes en programmation par contraintes - Théorie et application à des problèmes de recouvrement par des arbres*. Co-encadrement avec N. Beldiceanu.

– Sélection d'encadrement de projets de fin d'études (eq. master 2) à l'école des Mines de Nantes :

- Romain Megel (février à juillet 2010) : mise en œuvre d'un modèle de planification de campagne pour la régie publicitaire de TF1 dans la première version de l'outil de recherche locale *LocalSolver*, outil initialement développé et commercialisé par Bouygues SA ; depuis fin 2013, porté par la startUp Innovation 24. Encadrement entreprise : Férédric Gardy.

- Jean-Guillaume Fages (février à juillet 2011): Stage de recherche portant sur l'intégration d'un modèle de variables de graphe au sein de l'outil Choco. Un partie du travail réalisé autour d'un algorithme de filtrage a été publié à la conférence internationale CP'11.

- Germain Chabot (février à juillet 2012) : Sélection et mise en place d'un algorithme de planification amont pour optimiser le découpage des zones d'intervention et la répartition des ressources pour les équipes d'intervention d'ERDF. Encadrement entreprise : Mathieu Letortu.

- Philippe Lanoë (février à juillet 2012) : Optimisation de la consommation énergétique par l?analyse de données collectées d'un bâtiment contenant le Green Data Center d'IBM Montpellier. Le travail de ce stage a consisté en l'implémentation de modèles statistiques pour optimiser la consommation énergétique des bâtiments, ainsi que la préparation des données pour arriver à mettre en place ces modèles. Encadrement entreprise : Elsa Fabres et Saniya Ben Hassen-Marlin.

9.5. Responsabilités d'enseignements

– Sept. 2012 - auj. : responsable de l'UV "Informatique Décisionnelle" dans la formation par Apprentissage de l'École des Mines de Nantes (45h EDT, env. 20 élèves).

– Sept. 2011 - auj. : responsable de l'Option "Génie Informatique pour l'Aide à la Décision" de l'École des Mines de Nantes :

- responsable de l'UV "Graphe et Algorithmes" en 1ère année d'option (équiv. M1, 45h EDT, env. 20 élèves) ;

- responsable de l'UV "Architecture des Systèmes Informatiques et Logiciels" en 2nd année d'option (équiv. M2, 90h EDT, env. 20 élèves) ;

- responsable de l'UV "Informatique Décisionnelle" en 2nd année d'option (équiv. M2, 90h EDT, env. 20 élèves).

– **Sept. 2010 - auj. : responsable de l'UV "Projet d'Option"** en 2nd année d'option GIPAD de l'École des Mines de Nantes (équiv. M2, 90h EDT, env. 20 élèves).

– **Sept. 2008 - sept. 2011 : responsable du module "bases de données"** en cycle de base d'ingénieur de l'École des Mines de Nantes (45h EDT, env. 180 élèves).

9.6. Responsabilités collectives

– **Comités d'organisation de conférences :**

- CPAIOR'12, 9th Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (Nantes, 2012) ;

- JFPC'08, 4èmes Journées Francophones de la Programmation par Contraintes (Nantes, 2008) ;

- CP'06, 12th Conference on Principles and Practice of Constraint Programming (Nantes, 2006). En charge de l'organisation de la session poster présenté durant deux demi-journées lors de la conférence.

– **Déc. à juillet 2014 : co-animateur de la réflexion du rapprochement des départements Informatique et Automatique-productique**, École des Mines de Nantes.

– **Dès mars 2013 : pôle Images et Réseaux**, expert suppléant auprès du comité de sélection et de validation (Titulaire : Mario Südholt).

– **Janv. 2012 - auj. : membre nommé du conseil de laboratoire du LINA**, Laboratoire d'Informatique Nantes Atlantique.

– **Sept. 2008 - déc. 2011 : gestion du parc matériel du département informatique** de l'École des Mines de Nantes. En charge de la gestion du budget relatif à l'investissement en matériel informatique pour le département. Mes attributions allaient de la négociation budgétaire avec la direction pour l'année avenir, à la gestion des commandes pour le personnel du département.

– **2006 - 2007: représentant des doctorants**, département informatique de l'École des Mines.

– **2004 - 2006: secrétaire de l'association des doctorants** de l'École des Mines de Nantes, ACMIN.

9.7. Sélection de publications

De manière classique, le découpage des publications se fera selon trois catégories : les livres ou chapitres de livres, les revues internationales avec comité de sélection, les conférences internationales avec comité de sélection.

9.7.1. *Livres ou chapitres de livres*

[1] X. Lorca (2011). *Tree-based Graph Partitioning Constraint*. ISTE/Wiley, ISBN 978-1-84821-303-6. Version française : *Contraintes de Partitionnement de Graphe*. Hermès Sciences, collection *Programmation par Contraintes*, ISBN 978-2-746-24154-1 .

9.7.2. *Revues internationales avec comité de sélection*

[1] J.-G. Fages, X. Lorca, L.-M. Rousseau (2014). *The Salesman and the Tree: the importance of search in CP*, Constraints Journal. Accepted.

[2] C. Prud'Homme, X. Lorca, N. Jussien (2014). *Explanation-Based Large Neighborhood Search*, Constraints Journal. To appear.

[3] C. Prud'Homme, X. Lorca, N. Jussien, R. Douence (2014). *Propagation Engine Prototyping with a Domain Specific Language*, Constraints Journal. Vol. 19, issue 1, pages 57-76.

[4] N. Beldiceanu, I. Katriel, X. Lorca (2009). *Undirected Forest Constraints*, Annals of Operations Research 171(1): 127-147.

[5] N. Beldiceanu, P. Flener, X. Lorca (2008). *Combining Tree Partitioning, Precedence, and Incomparability Constraints*, Constraints Journal. Vol. 13, issue 4, pages 459-489.

9.7.3. *Conférences internationales avec comité de sélection*

[1] J.-G. Fages, X. Lorca, T. Petit (2014). *Self-decomposable Global Constraints*, in proceedings of the 21th European Conference on Artificial Intelligence, Prague, Czech Republic.

[2] J. Du Boisberranger, D. Gardy, X. Lorca, C. Truchet (2013). *When is it worthwhile to propagate a constraint? A probabilistic analysis of AllDifferent*, in proceedings of the meeting on Analytic Algorithmics and Combinatorics, New Orleans, Louisiana, USA.

[3] J.-G. Fages, X. Lorca (2011). *Revisiting the tree Constraint*, in proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, Perrugia, Italy.

[4] F. Hermenier, S. Demassey, X. Lorca (2011). *The Bin-Repacking Scheduling Problem in Virtualized Datacenters*, in proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, Perrugia, Italy.

[5] T. Petit, N. Beldiceanu, X. Lorca (2011). *A Generalized Arc-Consistency Algorithm for a Class of Counting Constraints*, in proceedings of the 22th International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain.

[6] N. Beldiceanu, F. Hermenier, X. Lorca, T. Petit (2010). *The increasing-nvalue Constraint*, in proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Bologna, Italy.

[7] G. Chabert, L. Jaulin, X. Lorca (2009). *A Constraint on the Number of Distinct Vectors with Application to Localization*, in proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, Lisbon, Portugal.

[8] F. Hermenier, X. Lorca, J-M. Menaud, G. Müller, J. Lawall (2009). *Entropy: a Consolidation Manager for Clusters*, in proceedings of the ACM International Conference on Virtual Execution Environments, Washington, DC, USA.

[9] R. Douence, X. Lorca, N. Loriant (2009). *Lazy Composition of Representations in Java*, in proceedings of the 8th International Conference on Software Composition, Zurich, Switzerland.

[10] N. Beldiceanu, X. Lorca (2007). *Necessary Condition for Path Partitioning Constraints*, in proceedings of the 4th International Conference on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, Brussels, Belgium.

[11] N. Beldiceanu, I. Katriel, X. Lorca (2006). *Undirected Forest Constraints*, in proceedings of the 3th International Conference on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, Cork, Ireland.

[12] N. Beldiceanu, P. Flener, X. Lorca (2005). *The* tree *Constraint*, in proceedings of the 2nd International Conference on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, Prague, Czech Republic.

Chapitre 10

Bibliographie

[ALS 99] ALSTRUP S., HAREL D., LAURIDSEN P., THORUP M., "Dominators in linear time", *SIAM J. Comput.*, vol. 28, n°6, p. 2117-2132, 1999.

[BAL 13] BALAFREJ A., BESSIERE C., COLETTA R., BOUYAKHF E.-L., "Adaptive Parameterized Consistency", *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Springer, p. 143-158, 2013.

[BEC 00] BECK J. C., PERRON L., "Discrepancy-Bounded Depth First Search", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, Lecture Notes in Computer Science, Springer, p. 8–10, 2000.

[BEL 01] BELDICEANU N., "Pruning for the Minimum Constraint Family and for the Number of Distinct Values Constraint Family", *Principles and Practice of Constraint Programming*, p. 211-224, 2001.

[BEL 05] BELDICEANU N., FLENER P., LORCA X., "The *tree* constraint", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, vol. 3524, p. 64-78, 2005.

[BEL 10a] BELDICEANU N., CARLSSON M., RAMPON J.-X., Global Constraint Catalog, 2nd Ed., Rapport n°T2010-07, SICS, 2010.

[BEL 10b] BELDICEANU N., HERMENIER F., LORCA X., PETIT T., "The *increasing nvalue* Constraint", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, vol. 6140 de *Lecture Notes in Computer Science*, Springer, p. 25–39, 2010.

[BEL 12] BELDICEANU N., SIMONIS H., "A Model Seeker: Extracting Global Constraint Models from Positive Examples", *Principles and Practice of Constraint Programming*, vol. 7514 de *Lecture Notes in Computer Science*, Springer, 2012.

[BEN 12] BENCHIMOL P., VAN HOEVE W. J., RÉGIN J.-C., ROUSSEAU L.-M., RUEHER M., "Improved filtering for weighted circuit constraints", *Constraints*, vol. 17, n°3, p. 205-233, 2012.

[BER 70] BERGE C., *Graphes et Hypergraphes*, Dunod, 1970.

[BES 94] BESSIÈRE C., "Arc-consistency and arc-consistency again", *Artif. Intell.*, vol. 65, n°1, p. 179-190, Elsevier Science Publishers Ltd., 1994.

[BES 95] BESSIÈRE C., FREUDER E. C., RÉGIN J.-C., "Using Inference to Reduce Arc Consistency Computation", *IJCAI*, p. 592-599, 1995.

[BES 96] BESSIÈRE C., RÉGIN J.-C., "MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems", FREUDER E. C., Ed., *Principles and Practice of Constraint Programming*, vol. 1118 de *Lecture Notes in Computer Science*, Springer, p. 61-75, 1996.

[BES 99] BESSIÈRE C., FREUDER E. C., RÉGIN J.-C., "Using Constraint Metaknowledge to Reduce Arc Consistency Computation", *Artif. Intell.*, vol. 107, n°1, p. 125-148, 1999.

[BES 03] BESSIÈRE C., VAN HENTENRYCK P., "To Be or Not to Be... a Global Constraint", *Principles and Practice of Constraint Programming*, p. 789-794, 2003.

[BES 04] BESSIÈRE C., HEBRARD E., HNICH B., WALSH T., "The Complexity of Global Constraints", *AAAI*, p. 112-117, 2004.

[BES 05a] BESSIÈRE C., HEBRARD E., HNICH B., KIZILTAN Z., WALSH T., "Filtering Algorithms for the NValue Constraint", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, vol. 3524 de *Lecture Notes in Computer Science*, Springer, p. 79-93, 2005.

[BES 05b] BESSIÈRE C., RÉGIN J.-C., YAP R. H. C., ZHANG Y., "An optimal coarse-grained arc consistency algorithm", *Artif. Intell.*, vol. 165, n°2, p. 165-185, 2005.

[BES 06] BESSIÈRE C., "Constraint Propagation", ROSSI F., VAN BEEK P., WALSH T., Eds., *Handbook of Constraint Programming*, Chapitre 3, Elsevier, 2006.

[BES 08a] BESSIÈRE C., HEBRARD E., HNICH B., KIZILTAN Z., WALSH T., "SLIDE: A useful special case of the CARDPATH constraint", *ECAI*, 2008.

[BES 08b] BESSIERE C., STERGIOU K., WALSH T., "Domain filtering consistencies for non-binary constraints", *Artif. Intell.*, vol. 172, n°6-7, p. 800-822, 2008.

[BES 13] BESSIERE C., COLETTA R., HEBRARD E., KATSIRELOS G., LAZAAR N., NARODYTSKA N., QUIMPER C.-G., WALSH T., "Constraint Acquisition via Partial Queries", *IJCAI*, 2013.

[BOI 11] DU BOISBERRANGER J., GARDY D., LORCA X., TRUCHET C., A Probabilistic Study of Bound Consistency for the Alldifferent Constraint, Rapport, Mines Nantes 11-01-INFO, 2011.

[BOI 13] BOISBERRANGER J. D., GARDY D., LORCA X., TRUCHET C., "When is it worthwhile to propagate a constraint? A probabilistic analysis of AllDifferent", *ANALCO*, p. 80-90, 2013.

[BOU 04] BOUSSEMART F., HEMERY F., LECOUTRE C., "Revision ordering heuristics for the Constraint Satisfaction Problem", *1st International Workshop on Constraint Propagation and Implementation held with CP'04(CPAI'04)*, Toronto, Canada, p. 9-43, sep 2004.

[BUC 98] BUCHSBAUM A., KAPLAN H., ROGERS A., WESTBROOK J., "A New, Simpler Linear-Time Dominators Algorithm", *ACM Transactions on Programming Languages and Systems*, vol. 20, p. 1265–1296, 1998.

[CAM 06] CAMBAZARD H., JUSSIEN N., "Identifying and exploiting problem structures using explanation-based constraint programming", *Constraints*, vol. 11, n°4, p. 295–313, Springer Verlag, 2006.

[CAR 13] CARLSONN M., SICStus Prolog, `http://sicstus.sics.se/`, 2013.

[CHM 98] CHMEISS A., JÉGOU P., "Efficient Path-Consistency Propagation", *International Journal on Artificial Intelligence Tools*, vol. 7, n°2, p. 121-142, 1998.

[COL 90] COLMERAUER A., "An Introduction to Prolog III", *Commun. ACM*, vol. 33, n°7, p. 69-90, 1990.

[Cos97] Cosytec, CHIP Reference Manual, release 5.1 édition, 1997.

[DAN 03] DANNA E., PERRON L., "Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs", *Principles and Practice of Constraint Programming*, vol. 2833 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 817-821, 2003.

[DEB 01] DEBRUYNE R., BESSIÈRE C., "Domain Filtering Consistencies", *J. Artif. Intell. Res. (JAIR)*, vol. 14, p. 205-230, 2001.

[DEM 06] DEMASSEY S., PESANT G., ROUSSEAU L.-M., "A Cost-Regular Based Hybrid Column Generation Approach", *Constraints*, vol. 11, n°4, p. 315-333, 2006.

[DEU 00] VAN DEURSEN A., KLINT P., VISSER J., "Domain-specific languages: an annotated bibliography", *SIGPLAN Not.*, vol. 35, n°6, p. 26–36, ACM Press, 2000.

[DOO 05] DOOMS G., DEVILLE Y., DUPONT P., "Cp(graph): Introducing a graph computation domain in constraint programming", *Principles and Practice of Constraint Programming*, vol. 3709 de *Lecture Notes in Computer Science*, Springer-Verlag, p. 211–225, 2005.

[FAG 11] FAGES J.-G., LORCA X., "Revisiting the tree Constraint", LEE J. H.-M., Ed., *Principles and Practice of Constraint Programming*, vol. 6876 de *Lecture Notes in Computer Science*, Springer, p. 271-285, 2011.

[FAG 12] FAGES J.-G., LORCA X., Improving the Asymmetric TSP by Considering Graph Structure, Rapport n°T12-4-INFO, EMN, 2012.

[FAG 13] FAGES J.-G., LORCA X., ROUSSEAU L.-M., "Une approche basée sur les contraintes pour résoudre le problème d'arbre recouvrant de coût minimum avec contraintes de degré", *Actes des 9èmes Journées Francophones de Programmation par Contraintes*, p. 119–122, 2013.

[FAG 14a] FAGES J.-G., LORCA X., PETIT T., "Self-decomposable Global Constraints", *ECAI*, Page to appear, 2014.

[FAG 14b] FAGES J.-G., LORCE X., ROUSSEAU L.-M., "The Salesman and the Tree: the importance of search in CP", *Constraints*, Pageto appear, 2014.

[FEL] FELDMAN J., "Java Specification Request 331: Constraint Programming API", `http://jcp.org/en/jsr/detail?id=331`, Accès: 13 Novembre 2013.

[G12 07] G12, "MiniZinc and FlatZinc", `http://www.minizinc.org/`, 2007, Accès: 12 Novembre 2013.

[GEN 06] GENT I. P., JEFFERSON C., MIGUEL I., "Watched Literals for Constraint Propagation in Minion", *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Springer, p. 182-197, 2006.

[GEN 08] GENT I. P., MIGUEL I., NIGHTINGALE P., "Generalised arc consistency for the AllDifferent constraint: An empirical survey", *Artif. Intell.*, vol. 172, n°18, p. 1973-2000, 2008.

[GEN 10] GENT I. P., MIGUEL I., MOORE N., "Lazy Explanations for Constraint Propagators", CARRO M., PEÑA R., Eds., *Practical Aspects of Declarative Languages*, vol. 5937 de *Lecture Notes in Computer Science*, p. 217-233, Springer Berlin Heidelberg, 2010.

[GON 85] GONDRAN M., MINOUX M., *Graphes et algorithmes*, Eyrolles, 2nd édition, 1985.

[HAR 79] HARALICK R. M., ELLIOTT G. L., "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", BUCHANAN B. G., Ed., *IJCAI*, William Kaufmann, p. 356-364, 1979.

[HAR 95] HARVEY W. D., GINSBERG M. L., "Limited Discrepancy Search", *IJCAI*, p. 607-615, 1995.

[HEL 04] HELLSTEN L., Consistency Propagation for *Stretch* Constraints, Master's thesis, Waterloo, 2004.

[HEN 89] VAN HENTENRYCK P., *Constraint satisfaction in logic programming*, MIT Press, 1989.

[HER 09] HERMENIER F., LORCA X., MENAUD J.-M., MULLER G., LAWALL J., "Entropy: a consolidation manager for clusters", *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, p. 41–50, 2009.

[HER 11] HERMENIER F., DEMASSEY S., LORCA X., "Bin Repacking Scheduling in Virtualized Datacenters", *Principles and Practice of Constraint Programming*, vol. 6876 de *Lecture Notes in Computer Science*, Springer, p. 27-41, 2011.

[HOE 01] VAN HOEVE W., "The alldifferent Constraint: A Survey", *CoRR*, vol. cs.PL/0105015, 2001.

[IBM 13] IBM, IBM CPLEX CP Optimizer, `http://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/index.html`, 2013.

[ITA 10] ITALIANO G. F., LAURA L., SANTARONI F., "Finding Strong Bridges and Strong Articulation Points in Linear Time", *International Conference on Combinatorial Optimization and Applications*, 2010.

[JUS 98] JUSSIEN N., LHOMME O., "Dynamic domain splitting for numeric CSPs", *European Conference on Artificial Intelligence (ECAI'98)*, p. 224–228, 1998.

[JUS 00a] JUSSIEN N., BARICHARD V., "The PaLM system: explanation-based constraint programming", *In Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, p. 118–133, 2000.

[JUS 00b] JUSSIEN N., DEBRUYNE R., BOIZUMAULT P., "Maintaining Arc-Consistency within Dynamic Backtracking", *Principles and Practice of Constraint Programming*, n° 1894Lecture Notes in Computer Science, Singapore, Springer-Verlag, p. 249–261, sep 2000.

[JUS 02] JUSSIEN N., LHOMME O., "Local search with constraint propagation and conflict-based heuristics", *Artificial Intelligence*, vol. 139, n°1, p. 21–45, Elsevier, jul 2002.

[JUS 03] JUSSIEN N., The versatility of using explanations within constraint programming, Rapport n°03-04-INFO, École des Mines de Nantes, 2003.

[KAT 06] KATRIEL I., "Expected-Case Analysis for Delayed Filtering", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, vol. 3990 de *Lecture Notes in Computer Science*, Springer, p. 119-125, 2006.

[KAT 12] KATSIRELOS G., NARODYTSKA N., WALSH T., "The SeqBin Constraint Revisited", *Principles and Practice of Constraint Programming*, vol. 7514 de *Lecture Notes in Computer Science*, Springer, p. 332-347, 2012.

[LAB 00] LABURTHE F., LE PROJET OCRE, "Choco : implémentation du noyau d'un système de contraintes", *Actes des 6e Journées Nationales sur la résolution de Problèmes NP-Complets*, p. 151–165, 2000.

[LAG 07] LAGERKVIST M. Z., SCHULTE C., "Advisors for Incremental Propagation.", *Principles and Practice of Constraint Programming*, vol. 4741 de *Lecture Notes in Computer Science*, Springer, p. 409-422, 2007.

[LAG 09] LAGERKVIST M. Z., SCHULTE C., "Propagator Groups.", *Principles and Practice of Constraint Programming*, vol. 5732 de *Lecture Notes in Computer Science*, Springer, p. 524-538, 2009.

[LAU 78] LAURIÈRE J.-L., "A Language and a Program for Stating and Solving Combinatorial Problems", *Artificial Intelligence*, vol. 10, p. 29-127, 1978.

[LEC 03] LECOUTRE C., BOUSSEMART F., HEMERY F., "Exploiting Multidirectionality in Coarse-Grained Arc Consistency Algorithms", *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Springer, p. 480-494, 2003.

[LEC 11] LECOUTRE C., HEMERY F., LECOUTRE C., SAMY-MODELIAR M., "Contrôle statistique du processus de propagation de contraintes", *7ièmes Journées Francophones de Programmation par Contraintes (JFPC'11)*, Lyon, France, p. 65-74, 2011.

[LOR 11] LORCA X., *Tree-based Graph Partitioning Constraint*, Wiley, 2011.

[MAC 77] MACKWORTH A., "Consistency in Networks of Relations.", *Artif. Intell.*, vol. 8, n°1, p. 99-118, 1977.

[MAC 85] MACKWORTH A., FREUDER E., "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", *Artif. Intell.*, vol. 25, n°1, p. 65-74, 1985.

[MCG 79] MCGREGOR J. J., "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms.", *Inf. Sci.*, vol. 19, n°3, p. 229-250, 1979.

[MEN 09] MENANA J., DEMASSEY S., "Sequencing and Counting with the multicost-regular Constraint", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, vol. 5547, p. 178-192, 2009.

[MIC 12] MICHEL L., HENTENRYCK P. V., "Activity-Based Search for Black-Box Constraint Programming Solvers", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, vol. 7298 de *Lecture Notes in Computer Science*, Springer, p. 228-243, 2012.

[MOH 86] MOHR R., HENDERSON T., "Arc and path consistency revised", *Artif. Intell.*, vol. 28, p. 225-233, 1986.

[MON 74] MONTANARI U., "Networks of constraints: Fundamental properties and applications to picture processing.", *Inf. Sci.*, vol. 7, p. 95-132, 1974.

[NET 07] NETHERCOTE N., STUCKEY P. J., BECKET R., BRAND S., DUCK G. J., TACK G., "MiniZinc: Towards a Standard CP Modelling Language", *Principles and Practice of Constraint Programming*, p. 529-543, 2007.

[OHR 09] OHRIMENKO O., STUCKEY P. J., CODISH M., "Propagation via lazy clause generation", *Constraints*, vol. 14, n°3, p. 357-391, Springer Verlag, 2009.

[PAC 99] PACHET F., ROY P., "Automatic Generation of Music Programs", *Principles and Practice of Constraint Programming*, vol. 1713 de *LNCS*, p. 331–345, 1999.

[PAP 02] PAPE C. L., PERRON L., RÉGIN J.-C., SHAW P., "Robust and Parallel Solving of a Network Design Problem", *Principles and Practice of Constraint Programming*, vol. 2470 de *Lecture Notes in Computer Science*, Springer, 2002.

[PAR 89] PARR T., "ANTLR v3", `http://www.antlr3.org/`, 1989, Accès: 12 Novembre 2013.

[PER 03] PERRON L., "Fast restart policies and large neighborhood search", *Principles and Practice of Constraint Programming*, vol. 2833 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2003.

[PER 04] PERRON L., SHAW P., FURNON V., "Propagation Guided Large Neighborhood Search", *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, p. 468-481, 2004.

[PES 01] PESANT G., "A Filtering Algorithm for the *Stretch* Constraint", *Principles and Practice of Constraint Programming*, vol. 2239 de *LNCS*, p. 183–195, 2001.

[PES 04] PESANT G., "A Regular Language Membership Constraint for Finite Sequences of Variables", *Principles and Practice of Constraint Programming*, vol. 3258, p. 482-495, 2004.

[PES 12] PESANT G., QUIMPER C.-G., ZANARINI A., "Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems", *J. Artif. Intell. Res. (JAIR)*, vol. 43, p. 173-210, 2012.

[PET 11] PETIT T., BELDICEANU N., LORCA X., "A Generalized Arc-Consistency Algorithm for a Class of Counting Constraints", WALSH T., Ed., *IJCAI*, p. 643-648, 2011.

[PIS 10] PISINGER D., ROPKE S., "Large neighborhood search", *Handbook of metaheuristics*, p. 399–419, Springer, 2010.

[PRU 13] PRUD'HOMME C., FAGES J.-G., "Introduction to Choco3", *1st Workshop on CPSolvers: Modeling, Applications, Integration, and Standardization, CP 2013*, http://choco.emn.fr, 2013.

[PRU 14a] PRUD'HOMME C., LORCA X., DOUENCE R., JUSSIEN N., "Propagation engine prototyping with a domain specific language", *Constraints*, vol. 19, n°1, p. 57-76, 2014.

[PRU 14b] PRUD'HOMME C., LORCA X., JUSSIEN N., "Explanation-Based Large Neighborhood Search", *Constraints*, Pageto appear, 2014.

[PUG 98] PUGET J.-F., "A fast algorithm for the bound consistency of alldiff constraints", *fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, p. 359-366, 1998.

[QUE 06] QUESADA L., Solving Constrained Graph Problems Using Reachability Constraints Based on Transitive Closure and Dominators, PhD thesis, Université catholique de Louvain, 2006.

[REF 04] REFALO P., "Impact-Based Search Strategies for Constraint Programming", *Principles and Practice of Constraint Programming*, vol. 3258 de *Lecture Notes in Computer Science*, Springer, p. 557-571, 2004.

[RÉG 94] RÉGIN J.-C., "A filtering algorithm for constraints of difference in CSP", *AAAI'94*, p. 362-367, 1994.

[RÉG 96] RÉGIN J.-C., "Generalized Arc Consistency for Global Cardinality Constraint", *AAAI'96*, p. 209-215, 1996.

[RÉG 04] RÉGIN J.-C., "CAC : Un algorithme d'arc-consistance configurable, générique et adaptatif", *JNPC*, Angers, France, 2004.

[RÉG 05] RÉGIN J.-C., "AC-*: A Configurable, Generic and Adaptive Arc Consistency Algorithm", *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Springer, p. 505-519, 2005.

[RÉG 08] RÉGIN J. C., "Simpler and incremental consistency checking and arc consistency filtering algorithm for the weighted tree constraint", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, vol. 5015, p. 233-247, 2008.

[ROS 06] ROSSI F., VAN BEEK P., WALSH T., *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., 2006.

[SCH 06] SCHULTE C., CARLSSON M., "Constraints in Procedural and Concurrent Languages", ROSSI F., VAN BEEK P., WALSH T., Eds., *Handbook of Constraint Programming*, Chapitre 14, Elsevier, 2006.

[SCH 08] SCHULTE C., STUCKEY P. J., "Efficient constraint propagation engines", *ACM Trans. Program. Lang. Syst.*, vol. 31, n°1, 2008.

[SCH 12] SCHRIJVERS T., TACK G., WUILLE P., SAMULOWITZ H., STUCKEY P. J., "Search Combinators", *CoRR*, vol. abs/1203.1095, 2012.

[SEL 03] SELLMANN M., "Cost-based filtering for shortest path constraints", *Principles and Practice of Constraint Programming*, vol. 2833, p. 694-708, 2003.

[SHA 98] SHAW P., "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems", *Principles and Practice of Constraint Programming*, vol. 1520 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 417-431, 1998.

[SMI 98] SMITH B. M., GRANT S. A., "Trying Harder to Fail First", *ECAI*, p. 249-253, 1998.

[SOR 04] SORLIN S., SOLNON C., "A global constraint for graph isomorphism problems", *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, vol. 3011, p. 287-301, 2004.

[TEA 13a] TEAM C., Choco Solver, `http://www.emn.fr/z-info/choco-solver/index.php?page=choco-3`, 2013.

[TEA 13b] TEAM G., Gecode Solver, `http://www.gecode.org/`, 2013.

[TEA 13c] TEAM J., JaCoP Solver, `http://jacop.osolpro.com/`, 2013.

[TEA 13d] TEAM M., Minion Solver, `http://minion.sourceforge.net/`, 2013.

[TEA 13e] TEAM O.-T., OR-Tools, `http://code.google.com/p/or-tools/`, 2013.

[Van 92] VAN HENTENRYCK P., DEVILLE Y., TENG C.-M., "A Generic Arc-Consistency Algorithm and its Specializations", *Artif. Intell.*, vol. 57, n°2-3, p. 291-321, 1992.

[VER 05] VERFAILLIE G., JUSSIEN N., "Constraint solving in uncertain and dynamic environments – a survey", *Constraints*, vol. 10, n°3, p. 253–281, Springer Verlag, 2005.

[WAL 97] WALSH T., "Depth-bounded Discrepancy Search", *IJCAI*, p. 1388-1395, 1997.

[ZAM 07] ZAMPELLI S., DEVILLES Y., SOLNON C., SORLIN S., DUPONT P., "Filtering for Subgraph Isomorphism", *Principles and Practice of Constraint Programming*, vol. 4741, p. 728-742, 2007.